2018

Improving Scalability of Evolutionary Robotics with Reformulation

Anton Bernatskiy
University of Vermont

Improving Scalability of Evolutionary Robotics with Reformulation

A Dissertation Presented

by

Anton Bernatskiy

to

The Faculty of the Graduate College

of

The University of Vermont

In Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
Specializing in Computer Science

October, 2018

Defense Date: August 20th, 2018
Dissertation Examination Committee:

Joshua C. Bongard, Ph.D., Advisor
Charles J. Goodnight, Ph.D., Chairperson
Margaret J. Eppstein, Ph.D.
Christopher M. Danforth, Ph.D.
Cynthia J. Forehand, Ph.D., Dean of Graduate College

Abstract

Creating systems that can operate autonomously in complex environments is a challenge for contemporary engineering techniques. Automatic design methods offer a promising alternative, but so far they have not been able to produce agents that outperform manual designs. One such method is evolutionary robotics. It has been shown to be a robust and versatile tool for designing robots to perform simple tasks, but more challenging tasks at present remain out of reach of the method.

In this thesis I discuss and attack some problems underlying the scalability issues associated with the method. I present a new technique for evolving modular networks. I show that the performance of modularity-biased evolution depends heavily on the morphology of the robot's body and present a new method for co-evolving morphology and modular control.

To be able to reason about the new technique I develop reformulation framework: a general way to describe and reason about metaoptimization approaches. Within this framework I describe a new heuristic for developing metaoptimization approaches that is based on the technique for co-evolving morphology and modularity. I validate the framework by applying it to a practical task of zero-g autonomous assembly of structures with a fleet of small robots.

Although this work focuses on the evolutionary robotics, methods and approaches developed within it can be applied to optimization problems in any domain.

Citations

Material from this dissertation has been published in the following form:

Bernatskiy, A., and Bongard, J.C.. (2015). "Exploiting the relationship between structural modularity and sparsity for faster network evolution." Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation.

AND

Bernatskiy, A., and Bongard, J.C.. (2017). "Choice of robot morphology can prohibit modular control and disrupt evolution." Proceedings of the 14th European Conference on Artificial Life. Vol. 14. P.60-67.

AND

Bernatskiy, A., and Bongard, J.C.. (2018). "Evolving morphology automatically reformulates the problem of designing modular control." Adaptive Behavior 26.2 (2018): 47-64.

Acknowledgements

Thanks to my advisor Josh Bongard for persistence in asking difficult questions and understanding difficult answers, cooperation and care. Thanks to Maggie Eppstein for keeping me grounded with her keen understanding of my work. Thanks to Chris Danforth for his many advices about the best mathematical tool ever - dynamical systems theory. Thanks to Charles Goodnight for jumping onboard when he was most needed there.

I thank my mother Tatiana and my sister Anna for giving me support when I most needed it. This work clearly wouldn't be possible without them. Special acknowlegement to our cat Kote for her grace which endlessly challenges the engineer in me.

Thanks to the Popov family - Roman, Mila and Daniel - for helping me survive my move to the USA, certain other events and generally just being around. Roman, Mila, this work would happen without you, but I would have way more gray hair by now.

Thanks to Eliza Guseva for being a US expert in the early days of this work and for the many scientific and philosophical discussions we had over the years.

Thanks to my friends and colleagues from UVM - Yu, Javier, David, Collin, Sam, Chris, Shawn, Zahra, Marcin and Misia, Shane, Jack and Ari. Special thanks to Mark for helping me navigate the formal side of my PhD program.

Thanks to our Vassar collaborators - Ken, John, Nick, Marc and Jody.

Thanks to Robert Snapp, Donna Rizzo and Ryan McDevitt for teaching outstanding, life-changing classes. Remote thanks to Hod Lipson, Robert Freitas, Ross Ashby, Gordon Pask and Xabier Barandiaran for blowing up my mind with their work.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

An agent is "anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators" [100]. In design of artificial agents one feature is almost always desirable: autonomy, the capacity to operate without human intervention for extended periods of time. Advances in design of autonomous artificial agents have profoundly reshaped humanity's way of life over past two centuries. Feedback regulators powered the huge leap of engineering thought during late nineteenth through the first half of twentieth century, bringing about modern heavy industry, transportation and early electronics. Autonomous spacecraft radically changed the way we gather geographical data, communicate and study the outer space. Introduction of industrial robots in 1950s revolutionized indoors manufacturing, making mass production cheaper than ever. Internet, the pinnacle of digital automation technology, is a layered cake made out of interacting autonomous agents, mostly artificial.

Despite such an impressive impact of agents engineering, autonomy has remained confined to simple environments for a long time. Feedback regulators rely on accu-

racy of models of systems being controlled and computational tractability of the corresponding control problems. Automatic spacecraft and aircraft exploit relative simplicity of laws governing their corresponding environments. Industrial robots mostly operate in controlled cells and rely very little on perceiving their environments. Most agents of the Internet operate in environments with few degrees of freedom and use well-defined protocols.

Recently, however, more complicated environments have been explored. In robotics, dynamically balanced legged locomotion[95, 64] and sharing the working space with humans[24, 99] has become possible. Practical artificial agents capable of processing human speech [72, 30] and raw sensory video streams [81, 4] have been developed. Stock market, despite being among the most sophisticated artificial environments, involves autonomous artificial agents to such extent that their behavior sometimes dominates the market dynamic [48].

Still, many of complex environments of practical interest remain out of reach of autonomous machines. Most operations in physical settings with complex structure, such as wilderness, excavation sites* and nuclear plant ruins, are performed either physically by humans or by robots that are meticulously controlled by humans. Autonomous space reconnaissance probes serve their purpose very well, but only until they have to interact with the complex structures found on the surface of other celestial bodies. At that point, as Steven Squyres, the principal investigator of NASA's Mars Exploration Rover Mission put it,

> [t]he unfortunate truth is that most things our rovers can do in a perfect sol [i.e. a martian day] a human explorer could do in less than a minute†

*With a notable exception of a few functional automated mining facilities [39, 123].

†[108], p.254-255, see also [27].

Economical benefits of achieving autonomy in such settings would be enormous. Obvious applications would yield buildings and power generating facilities at the cost close to that of the materials used; cheap excavation and mining; complete clean-up of nuclear disasters without human exposure to radiation. As J. Bongard put it, such robots would "do the same to outdoor production as what industrial robots did to the indoor production". However, those benefits are of minor importance compared to the benefits yielded by bolder applications.

Machine self-replication is arguably the most powerful constructive technology currently envisioned. Much like domestication of animals and plants, creation of such machines would enable humans to thrive in harsher environments, most notably outer space. With this technology the problem of energy generation can be solved, for all intents and purposes, forever [42]; enable the humans to efficiently mine, terraform and settle planets and their moons; eliminate the threat of asteroid impact; provide enough energy for manned interstellar spaceflight. At this point, machine replication in controlled, human designed environments has been demonstrated multiple times [41, 116, 130], The main hurdle on the way to constructing useful macroscopic replicators is the problem in question, autonomous operation in environments with complex structure.

For these reasons and others, a lot of effort has been made to improve our ability to create autonomous agents.

1.1 Challenges And Paradigms In Design Of Autonomous Agents

Design of agents for autonomous operation in environments with a complex structure poses a unique set of challenges. In this section I list some of these issues and compare the ways they are approached in the two existing AI paradigms: symbolic AI and nouvelle AI.

The first challenge is the large number of actions and interactions that are possible between a complex environment and an agent. Human designers are good at finding the ways to exploit some known interactions, but their sheer quantity and/or incomplete information about the environment can make manual design difficult. For example, it is possible to design an agent capable of some form of arboreal locomotion manually [84, 65]. However, the number of ways in which an arbitrary physical agent can interact with arboreal environment (coil around the trunks, brachiate on the branches, hang on leaf bundles etc, plus combinations) is large. The effort required to investigate all of them and come up with an optimal solution for every situation that agent might face is prohibitive.

Second, autonomous operation implies that the agent must survive, i.e. maintain its existence and functioning. In simple systems factors that may cause the agent to cease functioning can be figured out manually during design and testing phases. On the other hand, in complex systems some aspects of dynamics of the environment may exhibit unpredictable (chaotic or random) behavior, or even be unknown until certain actions are performed by the agent. As such, the agent must be able

to respond heuristically to some stimuli for which it was not explicitly designed to respond. Moreover, the response must be fast and environment-specific. For example, for a firefighter robot the heuristic survival strategy "move as fast as possible away from loud sounds and towards pre-identified strong structures capable of sheltering you" might work reasonably well. However, this approach will not work for a robot operating in environments in which ideas of "sound" or "move away" make little sense, such as in the vacuum of space or on a financial market.

The third challenge is a nontrivial combination of the first two. Aside from avoiding the immediate danger, it is desirable for the survival of an agent to actively maintain its functioning state against accumulating damage. Natural agents accomplish this with regeneration and reproduction. Little is known about how to reproduce these traits in artificial agents, but the complexity of the processes of growth in nature and manufacturing in engineering suggests that for complex environments the problem might be too hard to solve manually.

Traditionally, these issues are tackled by a top-down, manual design approach. A brute force approach to behavior generation, based on the sense-plan-act cycle and explicit representations of the environment, is employed. Known as **symbolic AI**, this approach has been utilized to design very good, but very expensive (in terms of research needed) solutions to some tasks in some complex environments, the most notable recent example being legged locomotion on rough terrain [95]. However, the range of tasks that can be solved by this approach remains limited by the availability of fast computation and human labor that can be spent on investigating the particularities of the task-environment pair.

The alternative approach known as **embodied** or **nouvelle AI**[16, 92]. It relies on

multi-scale, reactive approach to control and exploitation of the dynamical behavior of the agent-environment system. The approach is heavily inspired by our knowledge of living systems. Automatic design via evolutionary algorithms is often employed, an approach known as **evolutionary robotics**[88]. Connectionist representations and learning techniques are also often utilized (e.g. [106]).

Nouvelle AI has been successful in generating simple agents that learn intelligent behaviors from experience with minimal prior assumptions about the structure of the environment [106, 118]. It has produced some of the most impressive results in contemporary robotics in terms of resilience and behavior sophistication [15, 28, 74, 44]. Despite this, many of more complex environments remain out of reach. Although there is no single cause to this outcome, there is one major problem with the approach that is known to exacerbate many others: the large number of operations required to achieve reasonably good behaviors in evolutionary robotics [79, 103]. In the next section I review the knowledge that is currently available about causes of this phenomenon and existing approaches to combating it.

One of the key notions of nouvelle AI is **morphological computation** [55, 91, 92], the idea that given an environment and a task, complexity of control can be dramatically reduced if an appropriate morphology, or "body" is chosen. Morphology is an umbrella term for all aspects of the agent that are not control, i.e. those aspects that determine which aspects of the environment are perceived by the sensors, which are affected by motors and which are affected by the presence of the agent *per se*, without the involvement of the controller. For example, structure of human skeleton and mechanical properties of our muscles enable us to walk over flat ground with very minimal control that fits into out spinal cord.

1.2 Evolutionary Robotics

Evolutionary robotics is a technique for automatic creation of autonomous agents that relies on the Darwinian principle of reproduction of the fittest [88]. It is related to the fields of evolutionary computation and global optimization. Evolutionary computation is a broader field; it differs from evolutionary robotics in that it can work with non-agent entities [52, 35]. In the majority of cases an instance of evolutionary computation is also an instance of global optimization; however, research in global optimization is not tied to the population-based approach and is typically geared towards optimizing static functions. Due to these large overlaps, many issues and major ideas are common across the fields, which is reflected in citations.

In its most general form, evolutionary robotics approaches work as follows:

1. Generate an initial population of agents.
2. Directly or indirectly measure some metrics of the agent performance.
3. Select some subset of the agents for reproduction based on performance metrics.
4. Generate new agents by copying and combining the selected agents with random modifications.
5. Repeat steps 2-4 until some termination condition is met.

Details of each step vary by the approach.

The principal advantage of evolutionary robotics is its ability to automatically generate agents of almost any kind, with the only restriction being the designer's capacity to copy an agent while introducing some random change in it. The agents

may be represented as bit strings, graphs, generative programs or even evolved in hardware without a representation. It is also straightforward to combine representations for different aspects of the agent, such as its physical design (morphology) and control [105, 14, 20, 22]. This distinguishes evolutionary robotics from reinforcement learning [81, 74] where the learning algorithm is typically specific to the agent type (e.g. gradient descent for agents based on neural networks).

Many impressive autonomous agents and behaviors have been created with this technique [105, 44, 23, 69]. Some major results that were first obtained with this technique are automated generation of minimal cognitive behaviors [106] and creation of agents resiliently adapting to the environment through self-modeling [15, 28].

However, evolutionary robotics so far has not been able to scale up to more complex, and most importantly natural, environments. One major issue causing that is the large number of operations that are required to create an agent with this technique. While it is hard to identify all the causes of this issue, three large contributing factors can be isolated: interference between the aspects of the task, deception and bootstrap problem.

1.2.1 Interference Between The Aspects Of The Task And Modularity

This aspect is fully analogous to catastrophic forgetting in connectionist learning [43]. Generally, if some behavior has been learned by a population of robots under some selection pressure A, it may not be preserved once the selection pressure A is replaced by some other selection pressure B. And indeed, in practice it is often

the case that the older behavior is disrupted by the new adaptations. As a result, to create an agent capable of multiple behaviors it must be evaluated and selected for reproduction based on all of the behaviors. The situation becomes worse when the desirable behaviors are defined in a parametric way: the agent must be evaluated for each combination of parameters, with the number of evaluations growing exponentially with the number of parameters [79]. Complex environments have few symmetries, so just the number of distinct initial conditions makes the evaluation prohibitively expensive. For example, to evolve a robot to locomote to a light source placed on a patch of an uneven forest floor, the designer must evaluate its behavior for all starting points of the patch and, for each point, for all initial orientations.

Various approaches to this problem have been grouped under the term "modularity research". The idea is to create agents that have parts that encapsulate and protect different aspects of the behavior. To generate the behavior, those parts are mixed and matched. If the parts can be made to function independently on each other, then they can be evolved independently without one disrupting the other. An extension of this idea is that if the dependencies of the parts are, in some sense, "weak" (e.g. form a directed acyclic graph), then the parts can be optimized "almost" independently (e.g. in a topological order of the dependency graph).

The idea is widely used in engineering today [3] and arguably have been around since the Stone Age. The earliest mention of this idea in the context of the autonomous agent generation seems to be in Ross Ashby's "Design for a brain" [1].

The nature and operation of the parts depends on the type of the agent and can be multi-layered, with several qualitatively different modular decompositions coexisting within one agent type. Multiple kinds of modularity have been found in nature [122].

Among the kinds of modularity, *structural modularity* is one kind that particularly stands out in evolutionary theory and engineering. In a structurally modular system, the parts are the actual objects that are involved in the implementation of the agent. For example, if an agent consists of nodes and links, i.e. can be represented as a network, then it can be called modular if the network has more than one connected component or exhibits a community structure [45]. The heuristic behind this idea is that if a system can be decomposed into subsystems that are not connected *in the implementation*, then their evolution or learning can be done independently without one disrupting the other.

Many ways to evolve structural modularity have been proposed, both for disembodied [76, 56, 36, 26, 7] (*note - [7] is also provided as Section A.1*) and embodied settings [14]. Structural modularity was successfully used to combat the problem of the interference between the aspects of the task [34, 26] and, in the context of the evolution with sexual reproduction, even assist in the transfer of partial solutions within the population [68].

Some approaches to evolving structural modularity rely on designed modular variation of the task [56, 36], while others exploit statistical properties of networks to bias the evolution towards more modular solution ([26], see also Appendix A.1). All the disembodied approaches, however, rely on the assumption that the task-environment pair is solvable by a modular network in which the modules are arranged properly to assist with the task interference, an assumption that might not be true. Indeed, it has been shown that there is a disembodied task for which it is not the case ([26], figure S4).

Little is known, however, about how does this assumption and the associated

techniques interact with the embodied setting. It has been demonstrated that for some tasks, the evolution is more likely to produce more modular solution for some morphologies than for others [14]. The mechanism behind this phenomenon is, to some extent, elaborated upon in Chapter 2.

1.2.2 Deception And Multiple Timescales Of Learning

Global optimization is the area of research studying the methods for finding global optima of functions with multiple local optima (so-called multimodal functions). Many experiments in evolutionary robotics follow that paradigm, with some set of variables describing the agent being the inputs to the function and some experimenter-defined utility is the function being optimized.

If the function that is being optimized is a blackbox, the task of finding a global optimum is not solvable without exhaustively enumerating all possible combinations of input variables' values. That is why the performance of global optimization techniques, including the corresponding subset of evolutionary computation, is measured by their ability to find a "good-enough" solution: a solution that corresponds to a value that sufficiently close to the global optimum for practical purposes.

Deception in global optimization is any property of the optimized function that makes iterative optimization procedures prematurely converge to solutions that are worse than "good-enough". The points to which the evolution can converge prema-

turely can be local optima or saddles[‡], with the latter being far more common if the optimization occurs in a space with many dimensions [29].

Formal definition of the deception can be given as follows. Considering the optimization process as a discrete dynamical system, one can divide the space of possible solutions into basins of attraction. If basins of attraction corresponding to the solutions that are not "good-enough" have much larger hypervolume than their more hospitable counterparts, then the function (or a landscape) is said to be deceptive.

An example of a one-dimensional deceptive fitness landscape is given in fig. 1.1 (b).

Attacks on the problem of deception constitute a large fraction of all effort within the field of evolutionary computation. Several approaches exist:

Diversification. The selection algorithm is modified to explicitly or implicitly incentivise the escape from attractors, e.g. [46, 102, 94, 86]. The most extreme approach[71, 70] abandons the utility altogether, in favor of an efficient diversity objective.

Coevolution. The utility function is optimized alongside the agents, such that whenever the optimization settles on some solution, a parallel evolutionary process tries to come up with a different, typically harder task (e.g. [111]).

Constraining/biasing. The search is constrained to some subset of possible solutions, often by a nonlinear genotype-to-phenotype map that favors certain phenotypes. This approach works in much the same way as search patterns

[‡]Saddles can behave much like attractors if they have have many more attracting dimensions than repelling ones, or if the latter repel the points weakly. Any gradient-following optimization algorithm with a finite number of iterations will find such "attractors".

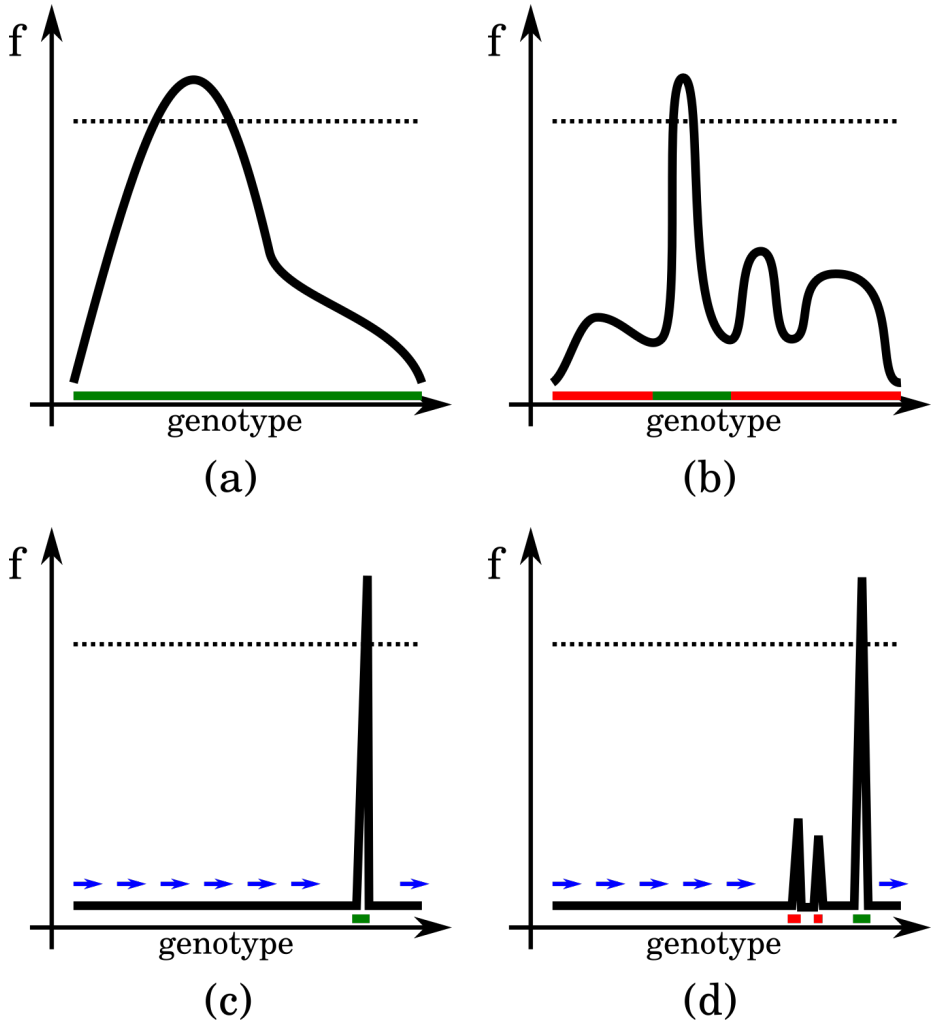


Figure 1.1: One-dimensional fitness functions f exhibiting properties typical for the fitness functions in evolutionary robotics. Dotted horizontal lines represent a "good-enough" level of fitness, red and green bars above the horizontal axis - attractors for gradient ascent. (a) Convex fitness function that can be optimized by following its gradient without much difficulty. (b) Deceptive fitness function: attractor for "good-enough" solution is much smaller than for the local optima. (c) Fitness function with a bootstrap problem: no gradient anywhere except for a tiny attractor of a "good-enough" solution. Blue arrows show the effect of one possible scaffolding: steady migration of the search to the right which helps find the solution. (d) Deceptive fitness function with a bootstrap problem. Solution space is mostly gradientless, with many tiny attractors that mostly lead to solutions that are not "good-enough". It can be seen that scaffolding does not help with deception in this example.

work in underwater searches [101, 67], by providing a guiding manifold that is easy to search with quasi-random search. If such a guide covers the search space densely enough so that it is likely to be an uninterrupted chain of beneficial mutations away from most potential optima, then at some point the search will happen upon the point that is close enough to a good optimum so that gradient climbing process can arrive to it without having to do any detrimental mutations. The most popular method of this family is [109]. Biasing works in a similar way, but instead of constraining the search to a subset of solution space the solutions that are closer, by some metric, to this subset, are visited earlier as the search algorithm iterates.

Nesting/metaoptimization. First discovered in the context of the relationship of learning over the lifetime and evolution, the idea now starts to take shape of the notion of synergy between different timescales of learning in which the evolution on one timescale speeds up the other by reducing the associated deceptiveness. The presence of a faster timescale increases the effective hypervolume of the genotypic space that an individual agent explores over its lifetime. This expands the sizes of all attractors, effectively smoothing the optimized function for the slower timescale.

If both timescales optimize the same subset of agent parameters, the result is a fast fixation of learned traits in the genotype, a phenomenon called **canalization**[120, 51].

Recently, it has been discovered that if the subset of the slowly optimized agent parameters represents a genotype-to-phenotype map and the fast timescale undergoes repeated periods of neutral and directional selection, then the slow

timescale evolves a map that biases the fast optimization away from the low fitness local optima[125, 61]. In effect, the fitness landscape for the fast optimization becomes smoother. Arguably, this finding constitutes the explanation of the elusive phenomenon of **evolution of evolvability**.

The ability to smooth fitness landscapes also makes certain forms of nesting useful in dealing with the bootstrap problem described in the next section.

In the embodied setting there is a natural separation of the aspects of the agent into morphology and control. If the control is governed by evolution, morphology can be seen as one aspect of genotype-to-phenotype map. Hence, the idea of evolution of evolvability readily applies if the morphology is optimized in a slower timescale than control. This observation is used by the emerging technique called **morphological protection**[21, 20].

Scaffolding. External knowledge is used to smooth the fitness landscape. For example, robots evolving for locomotion often use falling to achieve an initial increase of fitness. In this case genotypes that exhibit upright gaits are typically many deleterious mutations away from the genotypes that exhibit falling, making the falling strategy a deceptive local optimum. To alleviate this problem, in evolution of upright locomotion the upright posture is often explicitly rewarded or ensured by using external forces (e.g. [96]).

The knowledge used to smooth the function can be expressed as an additional objective prior to the optimization procedure, or learned online, e.g. from human observers [13, 8].

1.2.3 Bootstrap Problem And Scaffolding

In evolutionary robotics, designer's intent is expressed with utility function (error or fitness) that provide a quantitative measure of usefulness of a solution candidate. An empirical observation is that for many tasks and environments "natural", intuitively understandable utility functions have the same or almost the same value, corresponding to no progress towards the target behavior, within the vast majority of the solution space. Over these regions of the solution space there is no gradient that can be used by a gradient-climbing method (fig. 1.1 (c)), and such methods tend to perform on par or worse than random search on such landscapes. This is known as bootstrap problem [104].

This problem most often occurs in complex tasks that require coordination of several actions in time or space to make any progress. Here are some examples:

- Jumping, with height of the bottommost point of the robot's body as fitness. Consider an attempt to evolve a robot's body (composed of structural components, sensors and actuators) and controller to jump. Unless the actuators are very strong by default, the vast majority of robots and controllers in the search space never enter the fly phase and will have fitness of exactly zero.

- Manufacturing, with the amount of the final product as fitness. Consider the task of designing a system that transforms matter in stages (e.g. metabolic network). If in the evolutionary algorithm the genome rules how subsystems (e.g. individual chemical reactions) are combined to create such a system and more than one or two subsystems are required to produce the product, then vast majority of genotypes will not produce any and will have fitness of exactly

zero.

- Manipulation tasks in which the robot must travel to the object and carry it to some desired position. One measure of fitness is the negative difference between the initial and the desirable position. If there is any distance between the robot and the object at the beginning of the simulation, then most movements will not touch the object. The corresponding controllers will have all have the same value of fitness corresponding to the initial position of the object. More on this in chapter 4.

- Complex prey-catching strategies in prey-predator systems [47].

Bootstrap problem shares some features with deception, but these are different problems. Figure 1.1 (c) shows a non-deceptive fitness function that exhibits bootstrap problem: there is no gradient in most of the search space, but there is only one optimum and it leads to a "good-enough" solution. For a deceptive function with a bootstrap problem (fig. 1.1 (d)) there is also no gradient present, but there are local optima that do not lead to "good-enough" solutions. Figure 1.1 (d) shows the case when the landscape is deceptive, but has no bootstrap problem: there is gradient everywhere, yet the attractor for the "good-enough" maximum is much smaller than for the local optima.

Since the ratios between the hypervolumes of the "good-enough" attractor and the rest of the space falls exponentially as the number of dimensions grows, bootstrap problem and deception both become more difficult to deal with as the number of optimized variables increases.

When a high-dimensional fitness function exhibits the bootstrap problem, it will

very probably appear constant based on any reasonable sample of its values (i.e. a sample of a size much less than the size of the search space). The structure of the function thus remains invisible to blackbox optimization algorithms and no such algorithm can outperform random search [128].

All approaches to the bootstrap problem utilize some form of *a priori* knowledge to guide or augment the search. Since the bootstrap problem is similar to deception, all approaches that are useful in dealing with the bootstrap problem are also useful in dealing with deception, but not vice versa.

In **scaffolding** approaches, additional knowledge of potentially useful features of the final solution can be used to create a fitness gradient where there was none (fig. 1.1 (c),(d)). In the example of a manufacturing system evolution, that may correspond to adding extra fitness functions that reward production of intermediate products from which the final one can be produced with relative ease. There is a flurry of ways to define and combine these additional fitness functions. Approaches of this type include incremental evolution, behavioral decomposition and human-in-the-loop type approaches [104].

Temporal development is a type of metaoptimization that optimizes solutions that change over an additional fast timescale [51]. In evolutionary robotics this typically corresponds robots that change as they perform the task. If the task is such that the successful behavior can manifest itself for a fraction of the evaluation time and the fitness function rewards for that, then the number of evaluations is effectively increased by the factor of states that the agent visits during its lifetime. Behaviors that are hard to bootstrap, such as rolling, were evolved with this approach [62].

Diversity-based approaches can overcome the bootstrap problem by redefining

the task in terms of meaningful difference between genotypes (e.g. the evolution of upright gait in [70]).

1.3 Contribution Outline

Research described in this thesis began as an attempt to improve scalability of evolutionary robotics by combining the techniques of evolution of modularity and morphological computation via metaoptimization. In the process, however, more general principles were uncovered which can be applied to non-convex optimization outside of robotics.

By suppressing the interference between the aspects of the task, evolutionary techniques with a bias towards structural modularity have been shown to increase the convergence rate drastically [7, 26] (*note - [7] is also provided as Section A.1*). Indeed, in many cases the analogy with parallel and serial adaptation [1] applies, resulting in reduction of bounds for adaptation time from exponential to polynomial or even constant in the size of the problem. However, this speedup is only possible if the task can be solved by a structurally modular agent.

I demonstrate that morphology can determine whether or not the control task is solvable by a structurally modular controller (Chapter 2). I show that if the morphology is such that it enables modular control to solve the task, the techniques for solving modularity converge much more rapidly than for a morphology that does not admit modular control.

I show that this dependence of the rate of optimization of control on morphology can be used to evolve structurally modular agents (Chapter 3). This is achieved

by evolving the morphology alongside the control, but in a slower time scale, while applying the pressure towards structural modularity to control. In this case, structure of the control task varies together with morphology. Whenever it happens to admit a modular solution, the rate of convergence of the fast evolutionary timescale increases drastically, giving reproductive advantage to agents with the corresponding morphologies. If, additionally, morphologies that are few mutations away from each other tend to produce similar convergence rates, the landscape of morphological optimization becomes smooth and the convergence of the full evolutionary process can happen very fast and yield modular solutions. I show that the convergence in this case indeed happens at an increased rate, compared to the case when the morphology is not evolved.

I formulate an idea of **automatic reformulation** in optimization generalizes this method and many other existing ones (Chapter 3 and Section 5.1). Within this approach, design variables are separated into those that must be optimized as a part of solving the task (I call them **non-driving variables**) and those that influence the difficulty of this optimization process (**driving variables**). For example, in a task of automatic design of a neural network to approximate a certain mapping the weights of the neural network might selected as non-driving variables; then the numbers of hidden neurons and layers in the network might influence the difficulty of optimizing the weights, thus qualifying as driving variables. If difficulty of the optimization of non-driving variables varies strongly enough as the driving variables vary, then depending on the difficulty of optimizing the driving variables it might be advantageous, in terms of the overall optimization rate, to co-optimize driving and non-driving variables.

Based on my investigations of the morphology and modularity co-evolution I introduce a variant of the reformulation approach which I call **guided reformulation** (Chapter 3 and Section 5.1). It relies on the fact that the dependence of the difficulty of the optimization approach on certain variables is itself dependent on the biases of the underlying method for optimizing the non-driving variables. For example, in my investigations the dependence of the convergence rate of the control evolution on morphology was much more pronounced if the control evolution was biased towards structurally modular controllers. Thus, the impact of certain driving variables on the difficulty of the task can be increased by applying an appropriate bias to the optimization of the non-driving variables, making the automatic reformulation more efficient.

Formal treatment of the ideas of reformulation and guided reformulation is given in Section 5.1.

In Chapter 4 I validate the reformulation approach by using it to develop a new metaoptimization method to attack the bootstrap problem (through scaffolding) in the context of a practical problem of autonomous robotic assembly of structures in zero gravity. The new method can be used to tackle a vast array of optimization tasks suffering from the bootstrap problem.

Additionally, I introduce a new technique for evolving modularity (Appendix A.1). For a disembodied task I show that seeding the initial population with sparse networks substantially reduces the convergence time of the connection cost technique for evolving modularity [26]. Furthermore, I show that with this kind of initialization modular solutions can evolve in ordinary evolutionary algorithms without any selection pressure towards modularity. I observed an even more rapid convergence in this

case compared to the combination of this initialization and the connection cost technique, but unlike for this combination, modularity of the solutions that evolved over the initial period of rapid convergence later decayed. I widely used initial populations of sparse networks in the aforementioned investigations as an alternative to and and in conjunction with the performance and connection cost multiobjective optimization technique.

References for chapter 1

[1] W Ross Ashby. *Design for a Brain: The origin of adaptive behavior*. 2nd. London: Chapman & Hall Ltd, 1960.

[3] Carliss Y Baldwin and Kim B Clark. "Modularity in the design of complex engineering systems". In: *Complex engineered systems*. Springer, 2006, pp. 175–205.

[4] Andrea Banino et al. "Vector-based navigation using grid-like representations in artificial agents". In: *Nature* 557.7705 (2018), p. 429.

[7] Anton Bernatskiy and Joshua C Bongard. "Exploiting the relationship between structural modularity and sparsity for faster network evolution". In: *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM. 2015, pp. 1173–1176.

[8] Anton Bernatskiy, Gregory S Hornby, and Josh C Bongard. "Improving robot behavior optimization by combining user preferences". In: *Artificial Life* 14 (2014).

[13] Josh C Bongard and Gregory S Hornby. “Combining fitness-based search and user modeling in evolutionary robotics”. In: *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. ACM. 2013, pp. 159–166.

[14] Josh C Bongard et al. “Evolving robot morphology facilitates the evolution of neural modularity and evolvability”. In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM. 2015, pp. 129–136.

[15] Josh Bongard, Victor Zykov, and Hod Lipson. “Resilient machines through continuous self-modeling”. In: *Science* 314.5802 (2006), pp. 1118–1121.

[16] Rodney A Brooks. “Intelligence without representation”. In: *Artificial intelligence* 47.1 (1991), pp. 139–159.

[20] Nick Cheney et al. “On the Difficulty of Co-Optimizing Morphology and Control in Evolved Virtual Creatures”. In: *Proceedings of Artificial Life Conference*. 2016, pp. 226–234.

[21] Nick Cheney et al. “Scalable Co-Optimization of Morphology and Control in Embodied Machines”. In: *arXiv preprint arXiv:1706.06133* (2017).

[22] Nick Cheney et al. “Scalable co-optimization of morphology and control in embodied machines”. In: *Journal of The Royal Society Interface* 15.143 (2018), p. 20170937.

[23] Nick Cheney et al. “Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding”. In: *Proceedings of the*

15th annual conference on Genetic and evolutionary computation. ACM. 2013, pp. 167–174.

[24] Andrea Cherubini et al. "Collaborative manufacturing with physical human–robot interaction". In: *Robotics and Computer-Integrated Manufacturing* 40 (2016), pp. 1–13.

[26] Jeff Clune, Jean-Baptiste Mouret, and Hod Lipson. "The evolutionary origins of modularity". In: *Proc. R. Soc. B*. Vol. 280. 1755. The Royal Society. 2013, p. 20122863.

[27] Ian A Crawford. "Dispelling the myth of robotic efficiency". In: *Astronomy & Geophysics* 53.2 (2012), pp. 2–22.

[28] Antoine Cully et al. "Robots that can adapt like animals". In: *Nature* 521.7553 (2015), pp. 503–507.

[29] Yann N Dauphin et al. "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization". In: *Advances in neural information processing systems*. 2014, pp. 2933–2941.

[30] Ltsc Deng et al. "Recent advances in deep learning for speech research at Microsoft." In: *ICASSP*. Vol. 26. 2013, p. 64.

[34] Kai Olav Ellefsen, Jean-Baptiste Mouret, and Jeff Clune. "Neural modularity helps organisms evolve to learn new skills without forgetting old skills". In: *PLoS Comput Biol* 11.4 (2015), e1004128.

[35] Jacob A Englander and Bruce A Conway. "Automated Solution of the Low-Thrust Interplanetary Trajectory Problem". In: *Journal of Guidance, Control, and Dynamics* (2016), pp. 15–27.

[36] Carlos Espinosa-Soto and Andreas Wagner. “Specialization can drive the evolution of modularity”. In: *PLoS Comput Biol* 6.3 (2010), e1000719.

[39] Brian S Fisher and Sabine Schnittger. “Autonomous and remote operation technologies in the mining industry”. In: *BAeconomics Pty Ltd, February* (2012).

[41] Robert A Freitas Jr and Ralph C Merkle. *Kinematic self-replicating machines*. Landes Bioscience, 2004.

[42] Robert A Freitas and William P Gilbreath. “Advanced automation for space missions”. In: *Journal of the Astronautical Sciences* 30 (1982), pp. 1–11.

[43] Robert M French. “Catastrophic forgetting in connectionist networks”. In: *Trends in cognitive sciences* 3.4 (1999), pp. 128–135.

[44] Thomas Geijtenbeek, Michiel van de Panne, and A. Frank van der Stappen. “Flexible Muscle-Based Locomotion for Bipedal Creatures”. In: *ACM Transactions on Graphics* 32.6 (2013).

[45] Michelle Girvan and Mark EJ Newman. “Community structure in social and biological networks”. In: *Proceedings of the national academy of sciences* 99.12 (2002), pp. 7821–7826.

[46] David E Goldberg, Jon Richardson, et al. “Genetic algorithms with sharing for multimodal function optimization”. In: *Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms*. Hillsdale, NJ: Lawrence Erlbaum. 1987, pp. 41–49.

[47] Faustino Gomez and Risto Miikkulainen. “Incremental evolution of complex general behavior”. In: *Adaptive Behavior* 5.3-4 (1997), pp. 317–342.

[48] Joel Hasbrouck and Gideon Saar. “Low-latency trading”. In: *Journal of Financial Markets* 16.4 (2013), pp. 646–679.

[51] Geoffrey E Hinton and Steven J Nowlan. “How learning can guide evolution”. In: *Complex systems* 1.3 (1987), pp. 495–502.

[52] Gregory Hornby et al. “Automated antenna design with evolutionary algorithms”. In: *Space 2006*. 2015, p. 7242.

[55] Fumiya Iida, Raja Dravid, and Chandana Paul. “Design and control of a pendulum driven hopping robot”. In: *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*. Vol. 3. IEEE. 2002, pp. 2141–2146.

[56] Nadav Kashtan and Uri Alon. “Spontaneous evolution of modularity and network motifs”. In: *Proceedings of the National Academy of Sciences of the United States of America* 102.39 (2005), pp. 13773–13778.

[61] Kostas Kouvaris et al. “How evolution learns to generalise: Using the principles of learning theory to understand the evolution of developmental organisation”. In: *PLOS Computational Biology* 13.4 (2017), e1005358.

[62] Sam Kriegman, Nick Cheney, and Josh Bongard. “How morphological development can guide evolution”. In: *arXiv preprint arXiv:1711.07387* (2017).

[64] Scott Kuindersma et al. “Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot”. In: *Autonomous Robots* 40.3 (2016), pp. 429–455.

[65] Tin Lun Lam and Yangsheng Xu. “Biologically inspired tree-climbing robot with continuum maneuvering mechanism”. In: *Journal of Field Robotics* 29.6 (2012), pp. 843–860.

[67] Richard Larn and Rex Whistler. *Commercial diving manual*. David & Charles, 1993.

[68] Ari Larson et al. "Recombination Hotspots Promote the Evolvability of Modular Systems". In: *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*. ACM. 2016, pp. 115–116.

[69] Joel Lehman et al. "The surprising creativity of digital evolution: A collection of anecdotes from the evolutionary computation and artificial life research communities". In: *arXiv preprint arXiv:1803.03453* (2018).

[70] Joel Lehman and Kenneth O Stanley. "Abandoning objectives: Evolution through the search for novelty alone". In: *Evolutionary computation* 19.2 (2011), pp. 189–223.

[71] Joel Lehman and Kenneth O Stanley. "Exploiting Open-Endedness to Solve Problems Through the Search for Novelty." In: *ALIFE*. 2008, pp. 329–336.

[72] Jiwei Li et al. "Deep reinforcement learning for dialogue generation". In: *arXiv preprint arXiv:1606.01541* (2016).

[74] Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[76] Hod Lipson et al. "On the origin of modular variation". In: *Evolution* 56.8 (2002), pp. 1549–1556.

[79] Maja Matarić and Dave Cliff. "Challenges in evolving controllers for physical robots". In: *Robotics and autonomous systems* 19.1 (1996), pp. 67–83.

[81] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533.

[84] Jun Nakanishi, Toshio Fukuda, and Daniel E Koditschek. "A brachiating robot controller". In: *IEEE Transactions on Robotics and Automation* 16.2 (2000), pp. 109–123.

[86] Anh Mai Nguyen, Jason Yosinski, and Jeff Clune. "Innovation engines: Automated creativity and improved stochastic optimization via deep learning". In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM. 2015, pp. 959–966.

[88] Stefano Nolfi and Dario Floreano. *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*. MIT press, 2000.

[91] Chandana Paul. "Morphology and computation". In: *Proceedings of the International Conference on the Simulation of Adaptive Behaviour Los Angeles, CA, USA*. 2004, pp. 33–38.

[92] Rolf Pfeifer and Josh Bongard. *How the body shapes the way we think: a new view of intelligence*. MIT press, 2006.

[94] Kenneth Price, Rainer M Storn, and Jouni A Lampinen. *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media, 2006.

[95] Marc Raibert et al. "Bigdog, the rough-terrain quadruped robot". In: *IFAC Proceedings Volumes* 41.2 (2008), pp. 10822–10825.

[96] Torsten Reil and Phil Husbands. "Evolution of central pattern generators for bipedal walking in a real-time physics environment". In: *IEEE Transactions on Evolutionary Computation* 6.2 (2002), pp. 159–168.

[99] Leonel Rozo et al. “Learning physical collaborative robot behaviors from human demonstrations”. In: *IEEE Transactions on Robotics* 32.3 (2016), pp. 513–527.

[100] Stuart Jonathan Russell and Peter Norvig. *Artificial intelligence: a modern approach*. 3rd ed. Prentice Hall, 2009.

[101] Martin DJ Sayer. “NOAA Diving Manual: Diving for Science and Technology”. In: *Underwater Technology* 31.4 (2013), pp. 217–218.

[102] Michael Schmidt and Hod Lipson. “Age-fitness pareto optimization”. In: *Genetic Programming Theory and Practice VIII*. Springer, 2011, pp. 129–146.

[103] Fernando Silva, Luís Correia, and Anders Lyhne Christensen. “A case study on the scalability of online evolution of robotic controllers”. In: *Portuguese Conference on Artificial Intelligence*. Springer. 2015, pp. 189–200.

[104] Fernando Silva et al. “Open issues in evolutionary robotics”. In: *Evolutionary computation* 24.2 (2016), pp. 205–236.

[105] Karl Sims. “Evolving 3D morphology and behavior by competition”. In: *Artificial life* 1.4 (1994), pp. 353–372.

[106] Andrew C Slocum, Douglas C Downey, and Randall D Beer. “Further experiments in the evolution of minimally cognitive behavior: From perceiving affordances to selective attention”. In: *From animals to animats 6: Proceedings of the sixth international conference on simulation of adaptive behavior*. 2000, pp. 430–439.

[108] Steven Squyres. *Roving Mars: Spirit, Opportunity, and the exploration of the red planet*. Hachette Books, 2005.

[109] Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. "A hypercube-based encoding for evolving large-scale neural networks". In: *Artificial life* 15.2 (2009), pp. 185–212.

[111] Kenneth O Stanley and Risto Miikkulainen. "Competitive coevolution through evolutionary complexification". In: *J. Artif. Intell. Res.(JAIR)* 21 (2004), pp. 63–100.

[116] Jackrit Suthakorn, Andrew B Cushing, and Gregory S Chirikjian. "An autonomous self-replicating robotic system". In: *Proceedings of.* 2003, pp. 137–142.

[118] Elio Tuci, Gianluca Massera, and Stefano Nolfi. "Active categorical perception in an evolved anthropomorphic robotic arm". In: *Evolutionary Computation, 2009. CEC'09. IEEE Congress on.* IEEE. 2009, pp. 31–38.

[120] Conrad H Waddington. "Canalization of development and the inheritance of acquired characters". In: *Nature* 150.3811 (1942), pp. 563–565.

[122] Günter P Wagner, Mihaela Pavlicev, and James M Cheverud. "The road to modularity". In: *Nature Reviews Genetics* 8.12 (2007), pp. 921–931.

[123] Jinhua Wang and Zenghua Huang. "The Recent Technological Development of Intelligent Mining in China". In: *Engineering* 3.4 (2017), pp. 439–444.

[125] Richard A Watson and Eörs Szathmáry. "How can evolution learn?" In: *Trends in ecology & evolution* 31.2 (2016), pp. 147–157.

[128] David H Wolpert and William G Macready. "No free lunch theorems for optimization". In: *IEEE transactions on evolutionary computation* 1.1 (1997), pp. 67–82.

[130] Victor Zykov et al. “Robotics: Self-reproducing machines”. In: *Nature* 435.7039 (2005), pp. 163–164.

Chapter 2

Choice Of Robot Morphology Can Prohibit Modular Control And Disrupt Evolution

This work has been published as Bernatskiy, Anton, and Bongard, Josh C.. (2017). "Choice of robot morphology can prohibit modular control and disrupt evolution." Proceedings of the 14th European Conference on Artificial Life. Vol. 14. P.60-67.

2.1 Abstract

In evolutionary robotics, controllers are often represented as networks. Modularity is a desirable trait of such networks because modular networks are resistant to catastrophic forgetting and tend to have less connections than nonmodular ones. However, these advantages can only be realized if the control task is solvable by a modular network, and for any given practical task the control task depends on the

choice of the robot's morphology. Here we provide an example of a task solvable by robots with two different morphologies. We consider the most extreme kind of modularity – disconnectedness – and show that with the first morphology the task can be solved by a disconnected controller with few connections. On the other hand, the second morphology makes the task provably impossible for disconnected controllers and requires about three times more connections. For this morphology, most controllers that partially solve the task constitute local optima, forming an extremely deceptive fitness landscape. We show empirically that in this case a connection cost-based evolutionary algorithm for evolving modular controllers is greatly slowed down compared to the first morphology's case. Finally, this performance gap increases as the task is scaled up. These results show that the morphology may be a major factor determining the performance of controller optimization. Although in our task the optimal morphology is obvious to a human designer, we hypothesize that as evolutionary robotics is scaled to more sophisticated tasks the optimization of morphology alongside the control might become a requirement for evolving modular controllers.

2.2 Frequently used symbols

T_i – target orientation of ith segment;

A_i – absolute orientation of ith segment;

r_i – relative orientation of ith segment (defined in eq. (2.2));

s_i – reading of the target orientation sensor measuring the orientation of the ith segment (2.5);

f_i – motor output for ith segment (2.6);

$\boldsymbol{T}, \boldsymbol{A}, \boldsymbol{r}, \mathbf{s}, \mathbf{f}$ – corresponding N-dimensional vectors,

where

N – is a number of segments; J – sensor attachment matrix (2.5); K – constant $N \times N$ matrix such filled with ones on and below the main diagonal and zeros everywhere else (2.4).

2.3 Introduction

Evolutionary computation and particularly evolutionary robotics are important research tools in the area of artificial life ([66, 75, 97]). A lot of research effort concerning evolutionary computation is dedicated to the evolution of networks. Network representation has several advantages. First, it can describe many kinds of systems, including controllers and morphologies of artificial agents (e.g. [105]). Second, it is relatively straightforward to design genetic operators such as mutation and crossover for networks. Last but not least, a lot of models in biology are network-based, making it easier to draw inspiration from natural evolution.

One characteristic property of biological networks that attracts a lot of attention from evolutionary computation community is modularity (e.g. [45]). A network is structurally modular if its nodes can be divided into subsets (modules) that are connected more tightly within themselves than with the rest of the network. In computational (e.g. neural, genetic regulatory) networks structural modularity often leads to weak or absent functional dependence. Consequences of such weak dependence include resistance to catastrophic forgetting both in neuroevolutionary setting ([56, 36, 26]) and in learning ([34]). Such resistance allows for a reduced number of

training examples ([18]). In addition, modular networks tend to contain less connections, which further simplifies their optimization ([26, 7] (*note - [7] is also provided as Section A.1*)).

Although some techniques for evolving modular computational networks have been developed (e.g. [56, 36, 26, 7] (*note - [7] is also provided as Section A.1*)), they mostly focus on finding nearly-optimal modular solutions that are assumed to be exist. In addition, to harness the full power of the approach an appropriate modular variation of the task is desirable, either among the training examples ([18]) or over the evolutionary time ([56, 36, 26]). However, in practice, it cannot be assumed that modular solutions for a given task exist, nor that the task itself is modularly varying. Indeed, in the work presented here, we demonstrate that there is a robotic task and a robot morphology for which even the former assumption does not hold.

In evolutionary robotics, controllers are often represented as computational networks. Properties of modular networks make modularity a desirable property of such controllers. However, previous work ([14, 18]) suggests that performance of modular network evolution techniques in this setting can vary depending on the choice of the robot's body.

In [14] evolution produced more modular controllers if the morphology was under the evolutionary control. It was observed that certain morphologies enable modular control while others do not, but it was not clear which mechanism might be responsible for that.

In [18], a morphology is defined to be modular iff activation of less than all of robot's motors results in a change of less than all of its sensors. Similarly, a control system is modular iff a change in less than all of the sensors induces a change in less

than all of the motor neurons. It is shown that the number of environments in which the robot needs to be evaluated can be reduced significantly if both morphology and control are fixed to be modular.

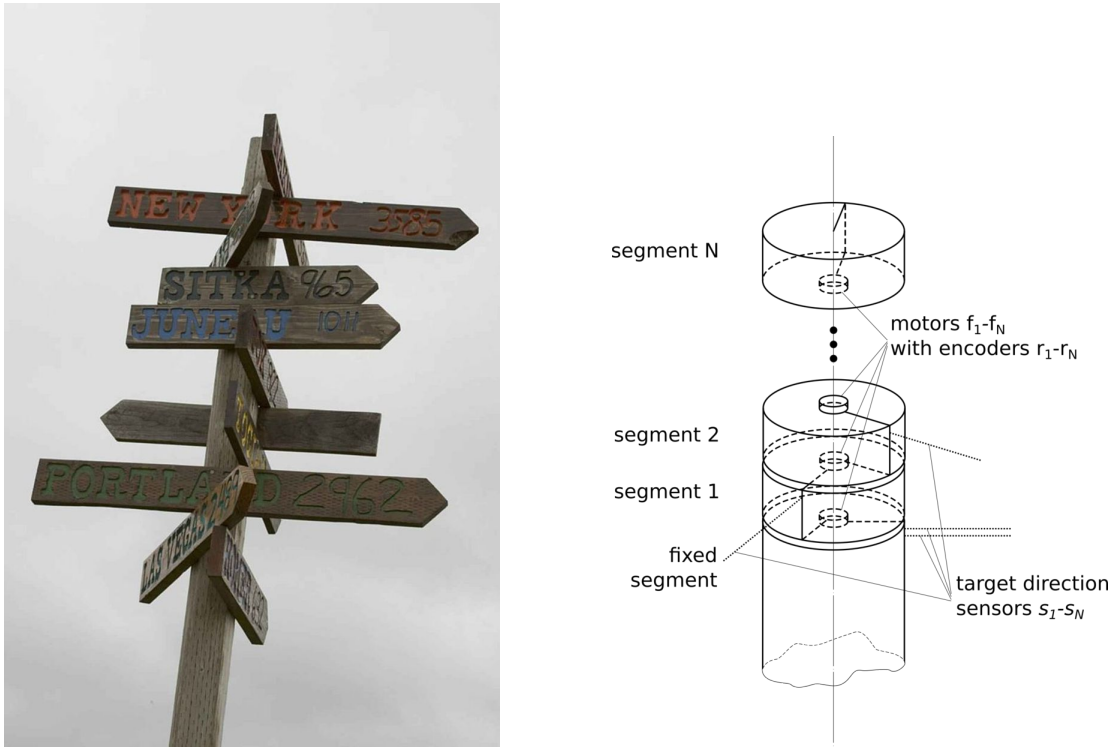
Despite these findings, many things remain unclear regarding the relationship between the morphology and modularity. In particular, while it is known that certain morphologies are beneficial for the evolution of modular controllers, it is not known why. Another open question is how much worse can the performance of evolution be if a less appropriate morphology is chosen. Here we fill these gaps by introducing a task and a family of robot morphologies with two extremities. The first is a morphology for which some optimal controllers consist of multiple disconnected modules. For the second morphology of interest it is provably impossible for any optimal controller to be disconnected. Additionally, we show the that the latter morphology induces an extremely deceptive fitness landscape in the space of possible controllers.

2.4 System description

2.4.1 Robot and task

Robots of the family described in this paper are called Arrowbots. An Arrowbot (Fig. 2.1b) is a dynamic version of road sign with multiple destinations (Fig. 2.1a). The task of a road sign is to show the direction towards several fixed objects, such as cities or mountains. In contrast, the task of an Arrowbot is to track N objects that may move and point towards them.

To accomplish this task the robot's body is divided into N segments attached to



a) b)

Figure 2.1: a) Regular road sign with multiple destinations. b) Arrowbot. Lines on the segments show orientations of segments' arrows.

each other in series by coaxial, actuated rotary joints, forming a stack (Fig. 2.1b). Segment #1 is located at the bottom of the stack. It is attached to a fixed base.

We define the orientation of the fixed base to be zero. For each segment i its absolute orientation angles A_i and target orientation T_i are defined relative to this reference (see Figure 2.2). If $A_i = T_i$, the segment points exactly in the target direction. Denoting $\boldsymbol{T} \equiv [T_1, T_2, ..., T_N]^T$, $\boldsymbol{A} \equiv [A_1, A_2, ..., A_N]^T$ we can reformulate the task as the minimization of

$$E \equiv |\boldsymbol{T} - \boldsymbol{A}|, \tag{2.1}$$

at $t \to \infty$ for some $\boldsymbol{T}(t)$ and initial condition $\boldsymbol{A}(t = 0)$. Throughout this work we assume constant target orientations, $\boldsymbol{T}(t) = const.$

Each segment i is associated with two sensors: a proprioceptive sensor and a target orientation sensor.

Each **proprioceptive sensor** measures the relative angle r_i between the orientation of its segment (ith) and the orientation of the $i-1$st segment below it (for segment #1, the fixed base). Readings of these sensors are tied to absolute orientations of segments:

$$\boldsymbol{A} = K\boldsymbol{r}, \tag{2.2}$$

or equivalently,

$$\boldsymbol{r} = K^{-1}\boldsymbol{A}. \tag{2.3}$$

Here $\boldsymbol{r} \equiv [r_1, r_2, ..., r_N]^T$ and K is defined to be the constant $N \times N$ matrix filled with

ones on and below its main diagonal and zeros above it:

$$K_{ij} = 1 \text{ if } i \geqslant j \text{ else } 0. \tag{2.4}$$

The determinant of this matrix is 1 for any N, so the inverse always exists, hence the equivalence of (2.2) and (2.3).

Each **target orientation sensor** s_i outputs the angle between the orientation of whatever it is attached to and the target orientation of its segment. Each target orientation sensor can be attached to any segment or to the fixed base.

Different ways of attaching target direction sensors give rise to different Arrowbot morphologies. We describe them with **sensor attachment matrix** J: an $N \times N$ matrix for which any element J_{ij} is equal to 1 if sensor s_i is attached to the jth segment and 0 otherwise. There is always exactly one sensor for every target direction, so every row of J contains at most one unit entry. If ith sensor is attached to the fixed base, then it is not attached to any of the moving segments and all the elements of the ith row of J are zeros.

With J we can express the absolute orientations of the parts to which the target orientation sensors are attached as $J\boldsymbol{A}$. Then target orientations sensor readings $\mathbf{s} \equiv [s_1, s_2, ..., s_N]$ are

$$\mathbf{s} = \boldsymbol{T} - J\boldsymbol{A} = \boldsymbol{T} - JK\boldsymbol{r}. \tag{2.5}$$

Actuated joints between the segments are the only motors of the system. Arrowbot's inputs are joint rotational velocities of segments relative to the segments right below them, $\dot{r_i}$.

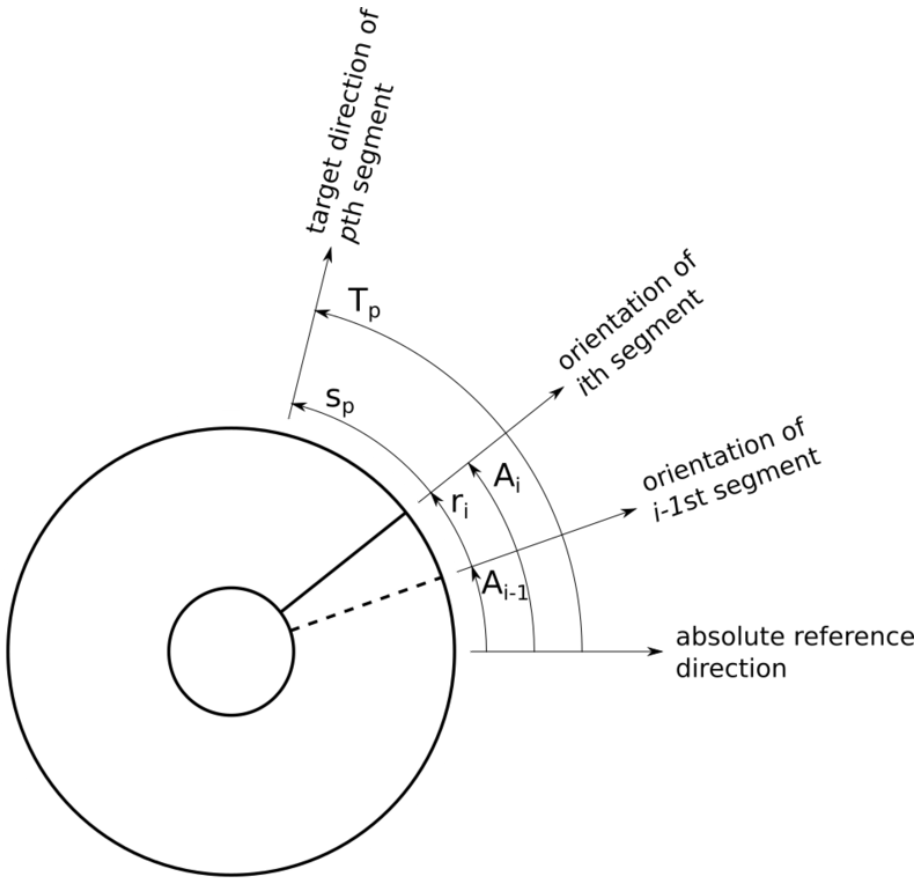


Figure 2.2: Arrowbot segment, top view. Solid radial line shows the orientation of the current (ith) segment. Dashed radial line shows the orientation of the segment right below the current one ($i-1st$). In this example, only one target direction sensor is attached to the segment. The sensor perceives the target direction of pth segment ($J_{pi}=1$).

2.5 Control

A controlled Arrowbot is described by the following dynamical system:

$$\dot{\boldsymbol{r}} = \mathbf{f}(\boldsymbol{r}, \mathbf{s}(\boldsymbol{r}, \boldsymbol{T})), \tag{2.6}$$

where $\mathbf{f}(\boldsymbol{r}, \mathbf{s}) : \mathbb{R}^{2N} \to \mathbb{R}^N$ is the controller and $\mathbf{s}(\boldsymbol{r}, \boldsymbol{T})$ is given by (2.5). For the controller to solve the task (2.1), this dynamical system must have an isolated, asymptotically stable fixed point corresponding to $\boldsymbol{T} = \boldsymbol{A} = K\boldsymbol{r}$. This translates into several conditions, each of which must be met for every $\boldsymbol{T} \in \mathbb{R}^N$. Equilibrium at $\boldsymbol{T} = K\boldsymbol{r}$ translates to:

$$\mathbf{f}(\boldsymbol{r} = K^{-1}\boldsymbol{T}, \mathbf{s}(\boldsymbol{r} = K^{-1}\boldsymbol{T}, \boldsymbol{T})) = \mathbf{0}. \tag{2.7}$$

The equilibrium point must be of the attracting, or asymptotically stable, kind:

$$\exists \delta_0 > 0 \,\text{s.t.}\, |K^{-1}\boldsymbol{T} - \boldsymbol{r}(0)| < \delta_0 \Rightarrow \lim_{t\to\infty} \boldsymbol{r}(t) \to K^{-1}\boldsymbol{T}, \tag{2.8}$$

where $\boldsymbol{r}(t)$ denotes the trajectory of the dynamical system (2.6) given target orientations $\boldsymbol{T}$ and initial conditions $\boldsymbol{r}(0)$.

If some controller, in addition to satisfying these two necessary conditions, also ensures that the fixed point is unique, then the point $\boldsymbol{A} = \boldsymbol{T}$ will attract all trajectories regardless of the initial conditions $\boldsymbol{r}(0)$. Such controllers are globally optimal for the task (2.1).

We characterize the connectivity of the controller using the following formalized notion of dependence. In a system with n variables $V = \{x_1, x_2, ..., x_n\}$ subject to some constraints C, x_i is **dependent** on x_j iff for some setting of the $n-2$ remaining variables $\hat{x}$ and some pair of settings $x_j' \neq x_j''$ the sets $X_i' \equiv \{x_i \,\text{such that}\, \{x_i, x_j', \hat{x}\} \,\text{satisfies}\, C\}$ and $X_i'' \equiv \{x_i \,\text{such that}\, \{x_i, x_j'', \hat{x}\} \,\text{satisfies}\, C\}$ do not coincide. Dependencies induced by the constraint C on the variables define an **undirected dependence graph** $G = (V, E)$ where $(x_i, x_j) \in E$ iff x_i depends on x_j or x_j depends on x_i.

If some variable x_i depends on some other variable x_j and the dependence is

not satisfied by definition of these variables (as is the case, for example, for radius and diameter or $\boldsymbol{r}$ and $\boldsymbol{A}$), then in any implementation of the constraint C x_i is connected to x_j via some kind of channel transmitting the information about x_j to the process generating x_i. For example, in a system involving two coordinates x, y of some material body on a plane the constraint $x = -y$ can only be enforced by introducing some physical contraption (e.g. a diagonal rail) which ensures that whenever y changes, x changes accordingly. Due to this property, the connectivity of the undirected dependence graph is the same as the connectivity of any undirected graph representing the information channels in the implementation, except possibly for situations when the implementation involves hidden variables that are neither influenced by nor influence the variables in V.

To investigate the connectivity of optimal Arrowbot controllers we consider undirected dependence graphs on $V^* = \{\mathbf{f}, \boldsymbol{r}, \mathbf{s}\}$ subject to the constraint $\mathbf{f} = \mathbf{f}(\boldsymbol{r}, \mathbf{s}(\boldsymbol{r}, \boldsymbol{T}))$. As we will see, necessary conditions (2.7) and (2.8) restrict the connectivity of the graphs in a way that depends on the sensor attachment matrix J, i.e. on the robot's morphology. Since the definitions of all variables in V^* do not imply any automatically satisfied constraints, we can draw conclusions about the connectivity of arbitrary nonlinear controllers from the connectivity of undirected dependence graphs.

This approach is inspired by similar tools used to treat dynamical dependencies as well as constraints: diagrams of immediate effect ([1]) and functional dependence graphs ([37]).

2.5.1 J=I ADMITS DISCONNECTED OPTIMAL CONTROLLERS

Attachment matrix $J = I$ corresponds to the morphology in which every sensor is attached to the segment for which it tracks the target orientation. In this case each sensor s_i measures its segment's signed pointing error, $T_i - A_i$.

Consider a family of controllers

$$\mathbf{f}(\boldsymbol{r}, \mathbf{s}) = W\mathbf{s}, \tag{2.9}$$

where $W = \text{diag}[w_{11}, w_{22}, ..., w_{NN}]$ are diagonal matrices for which all w_{ii} are positive. Then the dynamical system (2.6) turns into

$$\dot{\boldsymbol{r}} = W\boldsymbol{T} - WK\boldsymbol{r}. \tag{2.10}$$

Right hand side of this equation turns into 0 iff $\boldsymbol{T} = \boldsymbol{A} = Kr$, making sure that (2.7) and (2.15) are satisfied. Also, it shows that the fixed point $\boldsymbol{T} = \boldsymbol{A}$ is unique. The Jacobian

$$\begin{aligned} -WK &= \begin{bmatrix} -w_{11} & 0 & ... & 0 \\ 0 & -w_{22} & ... & 0 \\ & . & . & \\ 0 & 0 & ... & -w_{NN} \end{bmatrix} \begin{bmatrix} 1 & 0 & ... & 0 \\ 1 & 1 & ... & 0 \\ & & . \ . & \\ 1 & 1 & ... & 1 \end{bmatrix} = \\ &= \begin{bmatrix} -w_{11} & 0 & ... & 0 \\ -w_{22} & -w_{22} & ... & 0 \\ & . & . & \\ -w_{NN} & -w_{NN} & ... & -w_{NN} \end{bmatrix}. \end{aligned} \tag{2.11}$$

is a triangular matrix, therefore its eigenvalues are the values at the diagonal, $-w_{11}$, $-w_{22}$, ... , $-w_{NN}$. All of them are negative, so the stability condition (2.8) is also satisfied.

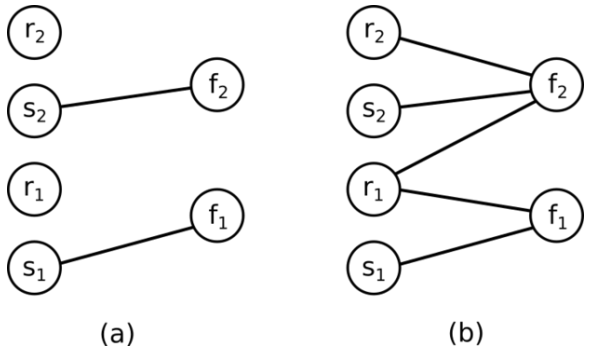


Figure 2.3: Examples of undirected dependency graphs (UDGs). (a) UDG of one of the controllers of the family (2.9) *optimal for $J = I$; (b) UDG of one possible optimal controller for $J = 0$.*

Therefore, all controllers of the family (2.9) are globally optimal for the task (2.1).

Every controller of this family is a disconnected network of $2N$ independent modules. Half of the modules connect motors $\dot{r}_i$ to target orientation sensors s_i associated with and attached to their segments. Another half are the proprioceptive sensors, which are, in these controllers, not connected to any other nodes.

2.5.2 There is no disconnected optimal controller for $J=0$

$J = 0$ corresponds to the case when all the target orientation sensors are attached to the fixed base of the robot and are directly measuring the target orientations, $\mathbf{s} = \boldsymbol{T}$. The dependence of $\mathbf{s}$ on $\boldsymbol{r}$ disappears, so in this case $\mathbf{s}$ can be treated as a constant vector of parameters. The dynamical system (2.6) then has a fixed point

whenever $\boldsymbol{r} = K^{-1}\mathbf{s}$:

$$\begin{aligned} &\forall \mathbf{s} \in \mathbb{R}^N \\ &[r_1, r_2, .., r_N] = [s_1, s_2 - s_1, .., s_N - s_{N-1}] \Rightarrow \\ &\mathbf{f}(\boldsymbol{r}, \mathbf{s}) = \mathbf{0} \end{aligned} \tag{2.12}$$

This also implies that for every point $\boldsymbol{r} \in \mathbb{R}^N$ there is a set of parameters $\mathbf{s} = K\boldsymbol{r}$ such that $\mathbf{f}(\boldsymbol{r}, \mathbf{s}) = \mathbf{0}$.

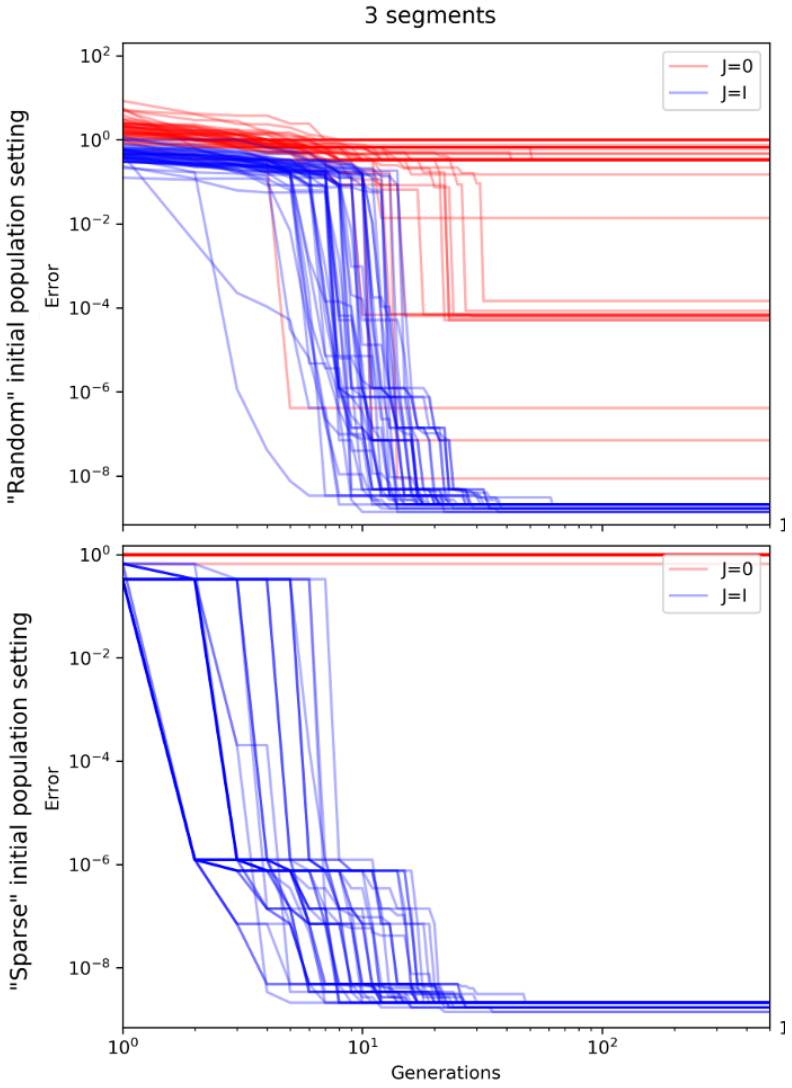
Theorem 1. Any controller $\mathbf{f}(\boldsymbol{r}, \mathbf{s})$ inducing an asymptotically stable fixed point in (2.6) for any $\mathbf{s} \in \mathbb{R}^N$ and $\boldsymbol{r} = K^{-1}\mathbf{s}$ has a connected undirected dependency graph.

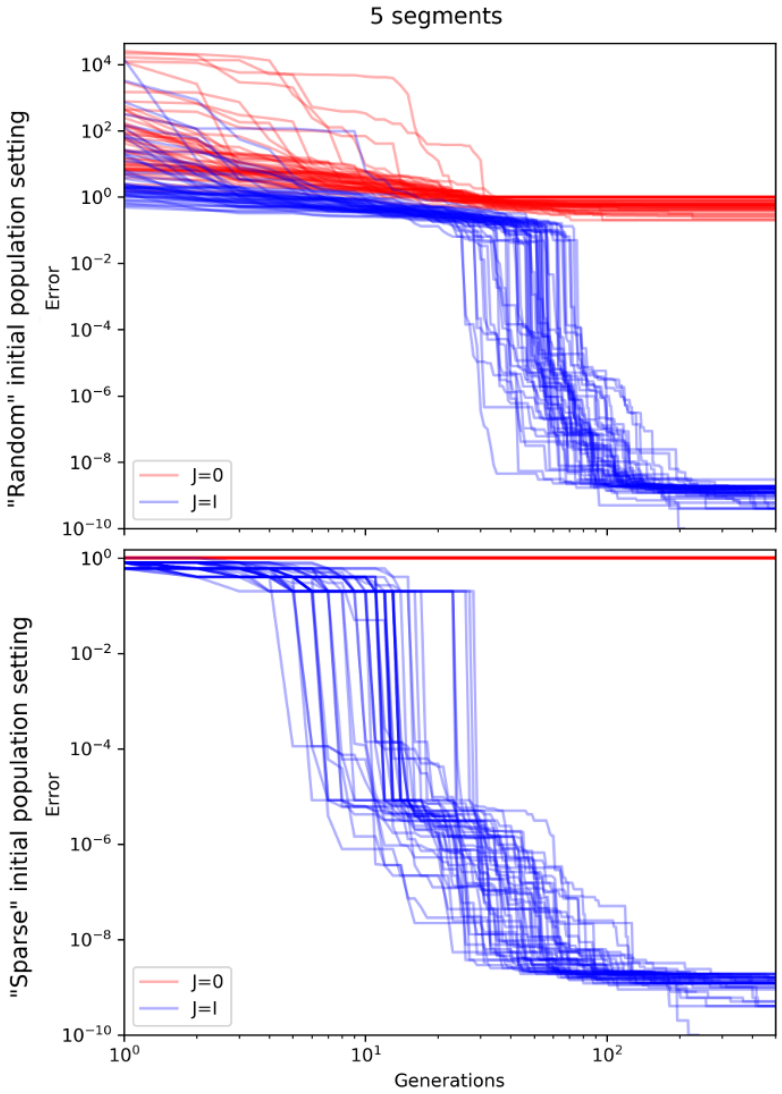
See Appendix A for the proof.

Theorem 1 implies that for $J = 0$ any optimal controller has only one connected component that is not isolated from sensors and motors. All sensors and motors, a total of $3N$ nodes, participate in this component. Such a component cannot be connected unless there are at least $3N - 1$, which is about three times more than what controllers for $J = 0$ require.

2.6 Evolution

We have shown that there are disconnected controllers among the globally optimal ones if the robot's morphology is defined by the sensor attachment matrix $J = I$ and that there are none if $J = 0$. However, it is not clear if this fact influences the complexity of optimizing the controller for the two morphologies, particularly if the optimization algorithm is designed to find modular solutions efficiently. We investigate this using a biobjective error-connection cost optimization [26].


3 segments
"Random" initial population setting
"Sparse" initial population setting
Error
Generations
J=0
J=I
10^{2}
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{1}


5 segments
"Random" initial population setting
"Sparse" initial population setting
Error
Generations
J=0
J=I

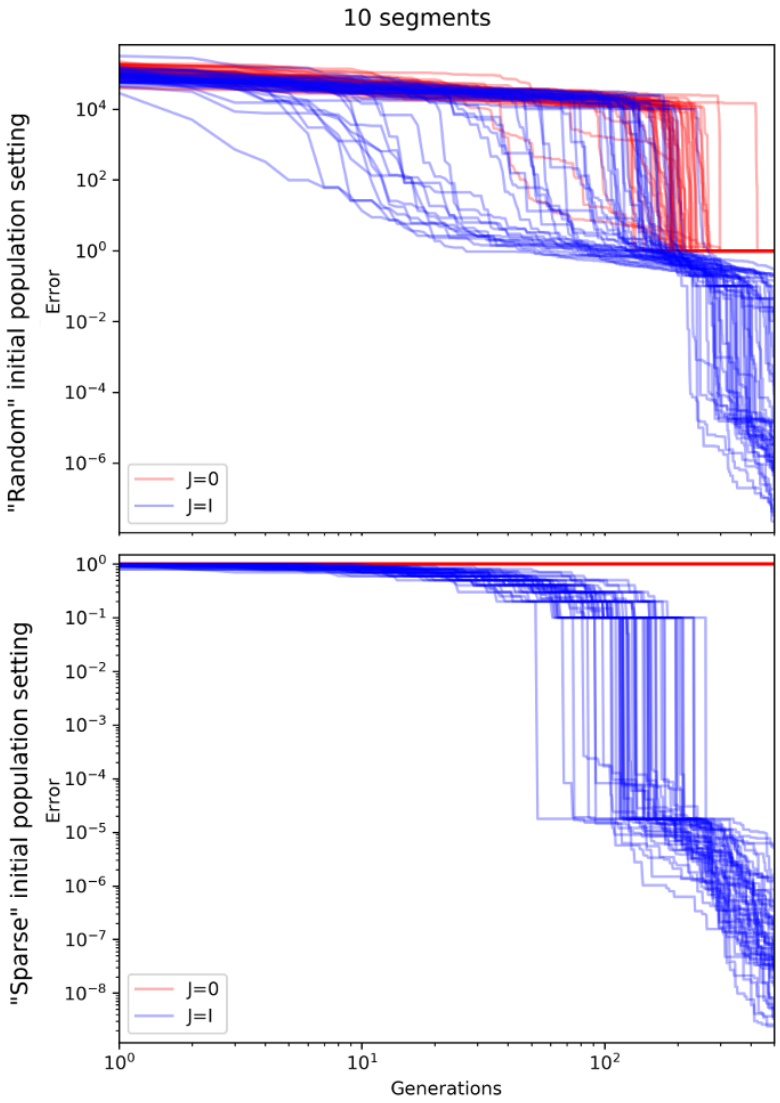


*Figure 2.2: Time series of the smallest error for evolution of Arrowbots with two different morphologies $J = 0$ and $J = I$. Columns (**each shown on a separate page in the dissertation version**) correspond to the three settings of Arrowbots' size (3,5 and 10 segments). For the runs in the top row, the evolution was initialized with a population of random networks; bottom row shows the performance if the initial population consists of sparse networks. Each of 50 trajectories is plotted in a semi-transparent line. Initial conditions in all cases are such that in every environment the error is initially equal to 1. It can be seen that for $J = 0$ evolution shows poor performance with random initial population and is completely disrupted for sparse initial population, even though for $J = I$ the optimum is found more rapidly in this setting.*

Each controller is represented by a vector of $2N^2$ connection weights encoding two $N \times N$ matrices (Y, W) such that

$$\dot{\boldsymbol{r}} = Y\boldsymbol{r} + W\mathbf{s}. \tag{2.13}$$

Connection weights can take values in $\{-1, 0, 1\}$.

Controllers are evaluated in a number of environments characterized by initial conditions $\boldsymbol{r}(t = 0)$ and target orientations $\boldsymbol{T}$. In each environment, each controller is evaluated by substituting (2.5) into (2.13) and integrating the resulting linear ODE system with fourth order Runge-Kutta method over a fixed time span $[0, 10]$ with a fixed timestep of 0.1. Pointing errors (2.1) are computed for each environment at $t = 10$ and averaged to obtain the final evaluation e.

Following the connection cost method for evolving modular networks [26], we simultaneously minimize pointing error and connection cost, defined as number of connections with nonzero weight. We use evolutionary algorithm identical to the one described in [7] (*note - also provided as Section A.1*). At each generation increment we select the pointing error – connection cost Pareto front of the current population and copy it into the new population. Then we proceed to add mutated copies of networks randomly chosen from Pareto front to the new population until it has the same size as the old one.

Mutation operator either (1) replaces a value of one nonzero weight with another (with probability of $p_1 = 0.5$ in all our experiments), or (2) adds a nonzero weight ($p_2 = 0.25$), or (3) removes a nonzero weight ($p_3 = 0.25$). If an impossible operation is attempted (e.g. nonzero weight has to be removed from an network with no such weights), the mutation is attempted repeatedly until it happens to perform a possible

operation.

Since the Pareto front is copied into the new population without modification regardless of its size, it can potentially fill the population completely and cause variation to cease. To prevent this, a sufficiently large population size must be chosen. Preliminary runs pointed to a population size of 50, and that was sufficient to avoid such variation cessation in any of the runs mentioned in this paper (see Table 2.1 for details).

We explored two types of initial populations – a population of networks generated by randomly choosing weights from $\{-1, 0, 1\}$ (**random** setup) and a population of networks generated by mutating an empty network once (**sparse** setup, [7] (*note - also provided as Section A.1*)). The latter setting trades off some initial variation to make the convergence faster if the task can be solved by a sparse and/or modular network. As we will see shortly, this leads to severe performance penalties if the task cannot be solved by such networks (in our case for $J = 0$).

2.7 Results and discussion

We investigated performance of the evolution for two types of initial populations (random and sparse) and three values of the number of segments parameter $N = 3, 5, 10$. Since each genome encoded a linear controller in form of two $N \times N$ matrices, the length of the genome has grown quadratically with N and the corresponding genome sizes were $18, 50, 200$. We ran batches of 50 evolutionary runs, 500 generations each, with populations of 50 individuals.

A set of 3 environments was used to evaluate controllers, with target orientations

$$\boldsymbol{T}^1 = [1, 0, 0], \boldsymbol{T}^2 = [0, 1, 0], \boldsymbol{T}^3 = [0, 0, 1]. \tag{2.14}$$

where the upper index indicates the number of the environment. Initial conditions were $\boldsymbol{r} = [0, 0, 0]$ in all three environments.

The results are shown in Figure 2.2 and 2.1. It can be seen that for $J = 0$ the performance of evolution is severely impaired, especially for the sparse initial population setting. In this case the error level is not improved relative to the initial conditions in any of 50 runs, while for $J = I$ a controller with near-zero error is found within 30 generations in all runs.

This result can be explained by considering the difference in response to mutation in non-optimal controllers between the two morphologies.

For $J = I$ the modules in the global optimum are disconnected and any controller that solves the task partially can retain the partial solution after mutation. This is a phenomenon called serial adaptation ([1]). In this case fitness landscape is convex.

The other morphology, $J = 0$, induces a more complicated landscape. Suppose, for this morphology, that at some point of the evolution there is a controller that successfully reduces the pointing error of the top $N - n$ segments of the Arrowbot, but not for the bottom n segments. Absolute orientation of nth segment A_n is required to compute $\dot{r}_i$ for $i = n + 1..N$. Since the task for the lower segments is not solved yet, $A_n = \sum_{i=1}^{n} r_i$ will have random dynamics, which must be exploited by the partial solution. Any mutation which improves the pointing error of any of the lower n segments will change this dynamics and likely break the partial solution. Thus, the fitness landscape is deceptive.

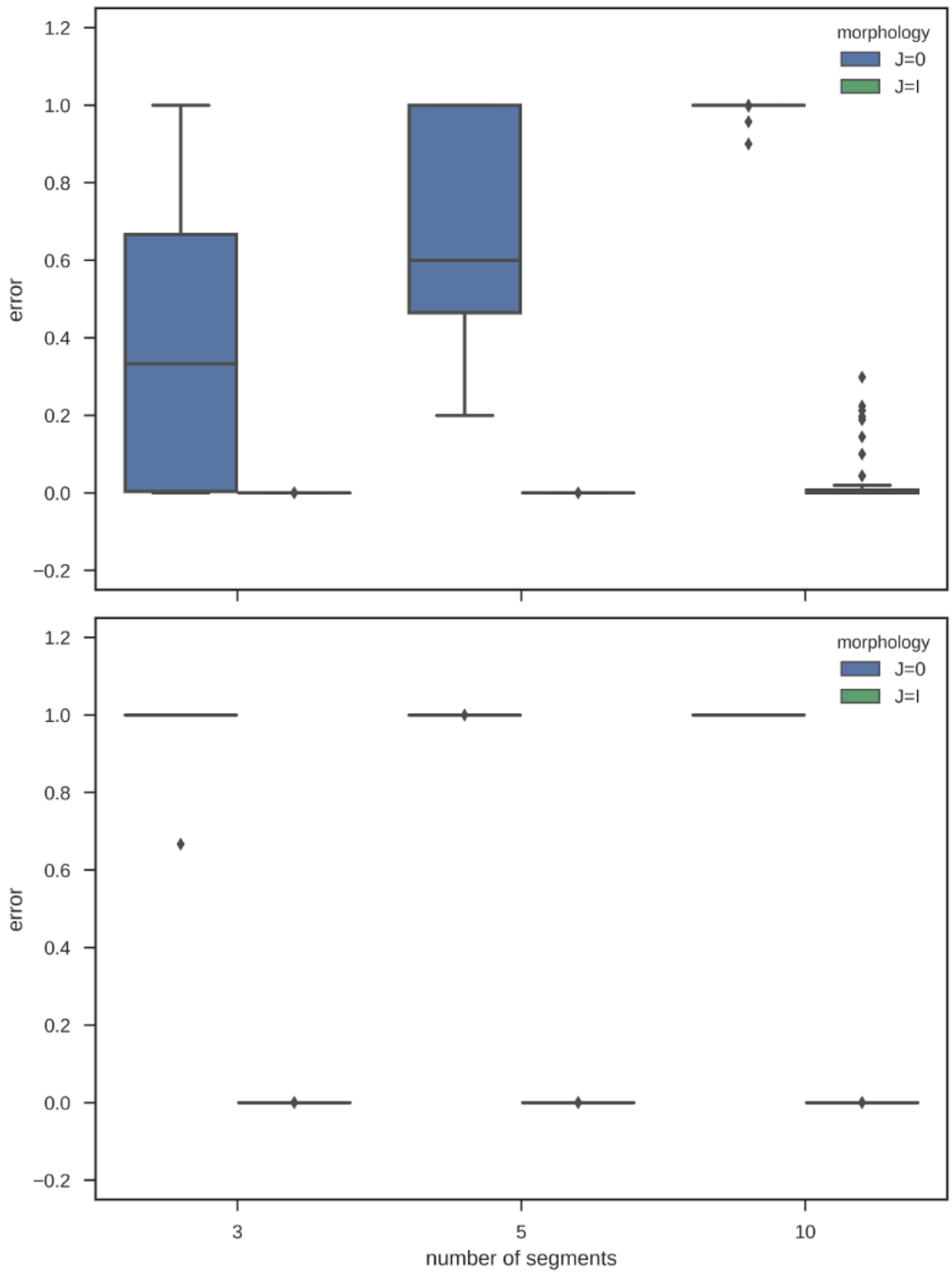


Figure 2.3: Final smallest errors for evolution of Arrowbots with different number of segments N after 500 generations. Top plot shows the performance if the evolution is initialized with a population of random networks; bottom plot shown the case of sparse initial population setting. It can be seen that the task becomes increasingly more challenging as N grows, especially for $J = 0$ and random initial population setting.

An initial population of sparse networks and the pressure to minimize the number of connections reinforce such deceptiveness. Under those conditions the most likely random behavior of the lower segments is staying at rest. This makes exploiting it especially simple and gives the local optima large basins of attraction with pronounced gradients, ultimately causing a complete disruption of the evolution.

Figure 2.3 shows how the performance of the evolution, measured by the final total square pointing error after 500 generations, varies with the number of segments N. Optimization task becomes more challenging as N grows. For the random initial population setting and $J = 0$, the evolution does not improve over the error of initial conditions at $N = 10$ in most runs, while for $J = I$ it does. For the sparse initial population setting, with $J = 0$ evolution does not improve in any but one of 3x50 runs, while for $J = I$ it achieves near zero pointing error in all 3x50 cases.

Initial population type	Number of segments		
	N=10	N=5	N=3
random	43/36.2	29/37.5	15/41.4
sparse	29/39.2	23/40.0	13/42.9

Table 2.1: Variation data for the evolutionary runs shown in Figure 2.2. Each cell shows maximum size of the error-connection cost Pareto front A across all generations and all runs and the average number of individuals mutated on each generation, B, in format A/B. Since the maximum Pareto front size never reaches the size of the population (50), the variation never ceases in any of the runs.

2.8 Conclusions

We have introduced Arrowbots, a task and a family of scalable robot morphologies which exhibits strong dependence of control modularity on morphology. In particular, we have shown that within this family there is (1) a morphology $J = I$ for which there are optimal controllers consisting of multiple disconnected modules and (2) a morphology $J = 0$ for which any optimal controller is necessarily connected. We have demonstrated that the performance of the evolution of Arrowbot control for the $J = 0$ morphology can be markedly worse than for the $J = 1$ and that the performance gap grows when Arrowbots with more segments are considered. We hypothesize that the difference in the performance of evolution is due to the extreme deceptiveness of the fitness landscape which arises if the robot has the $J = 0$ morphology and does not arise if $J = I$.

Thus, we have shown by construction that the choice of morphology can be the decisive factor in the evolution of modular controllers. The more aggressively the algorithm exploits the heuristic of modularity, the more important the morphology seems to become. Although for Arrowbots the morphology corresponding to most modular controllers is obvious to human designers, we hypothesize that in more sophisticated tasks this may not be the case. Optimization of the morphology alongside the control might be the solution of choice for those more sophisticated tasks.

2.9 Appendix A. Proof of Theorem 1.

Isolation: asymptotic stability (2.8) implies that the fixed point $\boldsymbol{r} = K^{-1}\boldsymbol{T}$ is isolated:

$$\exists \epsilon > 0 : 0 < |\boldsymbol{r} - K^{-1}\boldsymbol{T}| \leqslant \epsilon \Rightarrow \mathbf{f}(\boldsymbol{r}, \mathbf{s}(\boldsymbol{r}, \boldsymbol{T})) \neq \mathbf{0}. \tag{2.15}$$

The statement seems to be well known in the dynamical systems community, so I'll only provide a sketch of the proof. Suppose the opposite, then a fixed point $\boldsymbol{r}'$ can be found arbitrarily close to $K^{-1}\boldsymbol{T}$. Trajectories starting at $\boldsymbol{r}'$ do not approach $K^{-1}\boldsymbol{T}$, which contradict the asymptotic stability of that point.■

We begin by using this property, together with the asymptotic isolation itself, to constrain necessary dependencies of the involved variables. Then we show that under these constraints the undirected dependency network must be connected.

Lemma 1. In any controller $\mathbf{f}(\boldsymbol{r}, \mathbf{s})$ inducing a stable fixed point in (2.6) for any $\mathbf{s} \in \mathbb{R}^N$ and $\boldsymbol{r} = K^{-1}\mathbf{s}$ **(part 1)** any motor output $f_i(\boldsymbol{r}, \mathbf{s})$ depends on the readings of at least one proprioceptive sensor r_j and **(part 2)** for any proprioceptive sensor r_i, there is a motor output $f_j(\boldsymbol{r}, \mathbf{s})$ that depends on it.

Proof. Part 1: Suppose some motor output f_i is independent of all proprioceptive sensors $\boldsymbol{r}$. Then f_i is the same for all $\boldsymbol{r} \in \mathbb{R}^N$. Since we presupposed the existence of at least one fixed point, $f_i = 0$ at the point and therefore everywhere. It follows that $r_i = \text{const}$, which contradicts the asymptotic convergence.

Part 2: Suppose no motor output depends on some proprioceptive sensor r_i. For any $\mathbf{s}$ $\mathbf{f}(\boldsymbol{r}' \equiv K^{-1}\mathbf{s}, \mathbf{s}) = 0$. Now, consider a vector

$$\boldsymbol{r}'' \equiv K^{-1}\mathbf{s} + (0, .., \epsilon/2, .., 0) \tag{2.16}$$

where is the second term ϵ is at ith position in the vector. Since $\mathbf{f}$ is independent from r_i, $\mathbf{f}(\boldsymbol{r}'', \mathbf{s}) = 0$. Because ϵ can be chosen arbitrarily, that means that $\forall \epsilon > 0$ there is a fixed point $\boldsymbol{r}'' \neq \boldsymbol{r}'$ in the ϵ-neighborhood of $\boldsymbol{r}'$. This contradicts isolation. ■

Lemma 2. In any controller $\mathbf{f}(\boldsymbol{r}, \mathbf{s})$ inducing a stable fixed point in (2.6) at $\boldsymbol{T} = \boldsymbol{A}$ any target orientation sensor s has at least one motor output f that depends on it.

Proof. Suppose it is not, then there is a sensor s_i such that for any two values s_i and $s_i^* \in \mathbb{R}$ and any setting of remaining values $\hat{\mathbf{s}} \in \mathbb{R}^{N-1}$, $\boldsymbol{r} \in \mathbb{R}^N$ $\mathbf{f}(s_i, \hat{\mathbf{s}}, \boldsymbol{r}) = \mathbf{f}(s_i^*, \hat{\mathbf{s}}, \boldsymbol{r})$. Picking an arbitrary $\mathbf{s} \in \mathbb{R}^N$, we can choose its ith component as s_i, remaining values as $\hat{\mathbf{s}}$ and $K^{-1}\mathbf{s}$ as $\boldsymbol{r}$. By the conditions of Lemma 2, in this case the dynamical system (2.6) has a fixed point, so $\mathbf{f}(s_i, \hat{\mathbf{s}}, \boldsymbol{r}) = \mathbf{0}$.

We can also choose an arbitrary $\epsilon \in \mathbb{R}$ and set $s_i^* = s_i + \epsilon$, then

$$\mathbf{f}(s_i + \epsilon, \hat{\mathbf{s}}, \boldsymbol{r}) = \mathbf{f}(s_i, \hat{\mathbf{s}}, \boldsymbol{r}) = \mathbf{0}. \tag{2.17}$$

Denoting the vector $\mathbf{s}^*$ to have the same values as $\mathbf{s}$ except for $s_i^* = s_i + \epsilon$, we can conclude that the dynamical system $\dot{\boldsymbol{r}} = \mathbf{f}(\boldsymbol{r}, \mathbf{s}^*)$ has a fixed point at

$$\boldsymbol{r} = K^{-1}\mathbf{s} = [s_1, s_2 - s_1, ..., s_N - s_{N-1}]. \tag{2.18}$$

However, by Lemma conditions it also has an isolated fixed point at

$$\begin{aligned} \boldsymbol{r}^* \equiv K^{-1}\mathbf{s}^* = [&s_1, s_2 - s_1, ..., \\ &s_i + \epsilon - s_{i-1}, s_{i+1} - s_i - \epsilon, ..., s_N - s_{N-1}]. \end{aligned} \tag{2.19}$$

$\boldsymbol{r}$ and $\boldsymbol{r}^*$ are no further than 2ϵ from each other by any distance measure. By choosing appropriate $\epsilon = \epsilon'/3$, we can find a fixed point in any ϵ'-neighborhood of $\boldsymbol{r}^*$, which must be an isolated fixed point: contradiction. ■.

It follows from Lemma 1 that there is one-to-one correspondence between motors and proprioceptory sensors: they can be partitioned in N pairs (f_i, r_j) such that every f_i and every r_j participates in some pair and in each pair f_i depends on r_j. In each pair (f_i, r_j), f_i may also depend on some ss and rs other than r_j.

Theorem 1. Any controller $\mathbf{f}(\boldsymbol{r}, \mathbf{s})$ inducing an isolated, stable fixed point in (2.6) for any $\mathbf{s} \in \mathbb{R}^N$ and $\boldsymbol{r} = K^{-1}\mathbf{s}$ has a connected undirected dependencies graph.

Proof. Suppose there is a controller that has a disconnected undirected dependencies graph. In this case the variables $\mathbf{f}, \boldsymbol{r}, \mathbf{s}$ can be divided in two non-empty subsets α and β, such that no f in α depends on any r or s in β and vice versa. Let us denote the subset of all variables of type $B \in \{f, r, s\}$ in the subset $A \in \{\alpha, \beta\}$ as $\mathbf{B}^A$, e.g. a subset of all motors in α as $\mathbf{f}^\alpha$. Due to Lemma 1 each of the subsets will have an equal number m of motors f and proprioceptory sensors r: $|\mathbf{s}^A| = |\mathbf{f}^A| \equiv m^A$ for $A \in \{\alpha, \beta\}$. We will then have

$$m^\alpha + m^\beta = N. \tag{2.20}$$

By Lemma 2 neither subset can be composed only of proprioceptive sensors, so m^α and m^β are both greater than zero.

Since the motors in one set cannot depend on proprioceptive sensors in the other,

sets α and β will each form its own dynamical system:

$$\begin{aligned} \dot{\boldsymbol{r}^{\alpha}} &= \mathbf{f}^{\alpha}(\boldsymbol{r}^{\alpha}, \mathbf{s}^{\alpha}), \\ \dot{\boldsymbol{r}^{\beta}} &= \mathbf{f}^{\beta}(\boldsymbol{r}^{\beta}, \mathbf{s}^{\beta}). \end{aligned} \tag{2.21}$$

The systems are completely isolated, so the position of fixed points of each only depends on its set of parameters, $\mathbf{s}^{\alpha}$ for the first dynamical system and $\mathbf{s}^{\beta}$ for the second. Let us investigate the minimal sizes of these sets. One of the subsets α, β will contain r_1; let us pick α to be definite. In this case the fixed point condition for the α system is

$$\begin{aligned} & f^{\alpha}_{i_1}(r^{\alpha}_1 = s_1, ..., \mathbf{s}^{\alpha}) = 0, \\ & f^{\alpha}_{i_2}(r^{\alpha}_{j_2} = s_{j_2} - s_{j_2-1}, ..., \mathbf{s}^{\alpha}) = 0, \\ & ... \\ & f^{\alpha}_{i_{m^{\alpha}}}(r^{\alpha}_{j_{m^{\alpha}}} = s_{j_{m^{\alpha}}} - s_{j_{m^{\alpha}-1}}, ..., \mathbf{s}^{\alpha}) = 0. \end{aligned} \tag{2.22}$$

The number of parameters in $\mathbf{s}^{\alpha}$ should be at least the number of parameters through which $\boldsymbol{r}^{\alpha}$ is expressed; otherwise, due to the necessary dependence of fs on their corresponding rs, there will be a value of some s for which not all conditions (2.22) hold. Each condition adds at least one s into the expression for $\boldsymbol{r}^{\alpha}$, therefore

$$|\mathbf{s}^{\alpha}| \geqslant m^{\alpha}. \tag{2.23}$$

The other subset β does not contain r_1. Same reasoning applied to β leads to the

following conditions:

$$\begin{aligned} & f^{\beta}_{i_1}(r^{\beta}_{j_1} = s_{j_1} - s_{j_1-1}, ..., \mathbf{s}^{\beta}) = 0, \\ & ... \\ & f^{\beta}_{i_{m^\alpha}}(r^{\beta}_{j_{m^\beta}} = s_{j_{m^\beta}} - s_{j_{m^\beta}-1}, ..., \mathbf{s}^{\beta}) = 0. \end{aligned} \tag{2.24}$$

The first equation adds at least two proprioceptory sensors into the expression for $\boldsymbol{r}^{\beta}$ at the fixed point, and each subsequent condition adds at least one. This gives

$$|\mathbf{s}^{\beta}| \geqslant 1 + m^{\beta}. \tag{2.25}$$

Since $\mathbf{s}^{\alpha} \cap \mathbf{s}^{\beta} = \varnothing$ and $\mathbf{s}^{\alpha} \cup \mathbf{s}^{\beta} = \mathbf{s}$, equations (2.23), (2.25) and (2.20) can be combined to yield

$$|\mathbf{s}| = N \geqslant m^{\alpha} + m^{\beta} + 1 = N + 1. \tag{2.26}$$

Contradiction. ■

References for chapter 2

[1] W Ross Ashby. *Design for a Brain: The origin of adaptive behavior*. 2nd. London: Chapman & Hall Ltd, 1960.

[7] Anton Bernatskiy and Joshua C Bongard. "Exploiting the relationship between structural modularity and sparsity for faster network evolution". In: *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM. 2015, pp. 1173–1176.

[14] Josh C Bongard et al. “Evolving robot morphology facilitates the evolution of neural modularity and evolvability”. In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM. 2015, pp. 129–136.

[18] Collin K Cappelle et al. “Morphological Modularity Can Enable the Evolution of Robot Behavior to Scale Linearly with the Number of Environmental Features”. In: *Frontiers in Robotics and AI* 3 (2016), p. 59.

[26] Jeff Clune, Jean-Baptiste Mouret, and Hod Lipson. “The evolutionary origins of modularity”. In: *Proc. R. Soc. B*. Vol. 280. 1755. The Royal Society. 2013, p. 20122863.

[34] Kai Olav Ellefsen, Jean-Baptiste Mouret, and Jeff Clune. “Neural modularity helps organisms evolve to learn new skills without forgetting old skills”. In: *PLoS Comput Biol* 11.4 (2015), e1004128.

[36] Carlos Espinosa-Soto and Andreas Wagner. “Specialization can drive the evolution of modularity”. In: *PLoS Comput Biol* 6.3 (2010), e1000719.

[37] Jalal Etesami and Negar Kiyavash. “Measuring Causal Relationships in Dynamical Systems through Recovery of Functional Dependencies”. In: *IEEE Transactions on Signal and Information Processing over Networks* (2016).

[45] Michelle Girvan and Mark EJ Newman. “Community structure in social and biological networks”. In: *Proceedings of the national academy of sciences* 99.12 (2002), pp. 7821–7826.

[56] Nadav Kashtan and Uri Alon. “Spontaneous evolution of modularity and network motifs”. In: *Proceedings of the National Academy of Sciences of the United States of America* 102.39 (2005), pp. 13773–13778.

[66] Christopher G Langton. *Artificial life: An overview*. Mit Press, 1997.

[75] Hod Lipson and Jordan B Pollack. “Automatic design and manufacture of robotic lifeforms”. In: *Nature* 406.6799 (2000), pp. 974–978.

[97] Marieke Rohde. *Enaction, embodiment, evolutionary robotics: simulation models for a post-cognitivist science of mind*. Vol. 1. Springer Science & Business Media, 2010.

[105] Karl Sims. “Evolving 3D morphology and behavior by competition”. In: *Artificial life* 1.4 (1994), pp. 353–372.

Chapter 3

Evolving Morphology Automatically Reformulates The Problem Of Designing Modular Control

This work has been published as Bernatskiy, Anton, and Bongard, Josh C.. (2018). "Evolving morphology automatically reformulates the problem of designing modular control." Adaptive Behavior 26.2 (2018): 47-64.

3.1 Abstract

Modularity is a system property of many natural and artificial adaptive systems. Evolutionary algorithms designed to produce modular solutions have increased convergence rates and improved generalization ability; however, their performance can

be impacted if the task is inherently nonmodular. Previously we have shown that some design variables can influence whether the task on the remaining variables is inherently modular. We investigate the possibility of exploiting that dependence to simplify optimization and arrive at a general design pattern that we use to show that evolutionary search can seek such modularity-inducing design variable values, thus easing subsequent search for highly fit, modular organization within the remaining design variables. We investigate this approach with embodied agents in which evolutionary discovery of morphology enables subsequent discovery of highly fit, modular controllers and show that it benefits from biasing search toward modular controllers and setting the mutation rate for control policies higher than that for morphology. This work also reinforces our previous finding that the relationship between modularity and evolvability that is well studied in nonembodied systems can, under certain conditions, be generalized to include embodied systems as well, and provides a practical approach to satisfying the conditions in question.

3.2 Frequently Used Symbols

N – the number of segments;

T_i – target orientation of ith segment;

A_i – absolute orientation of ith segment;

r_i – relative orientation of ith segment (see equation (3.1));

s_i – reading of the target orientation sensor measuring the orientation of the ith segment relative to the body the sensor is attached to (see equation (3.3));

m_i – motor output for ith segment (see equation (3.5));

$\boldsymbol{T}, \boldsymbol{A}, \boldsymbol{r}, \mathbf{s}, \mathbf{f}$ – corresponding N-dimensional vectors;

J – $N \times N$ target orientation sensor attachment matrix (see equation (3.3));

K – constant $N \times N$ matrix such filled with ones on and below the main diagonal and zeros everywhere else (see equation (3.2));

W, Y – matrices of coefficients of a linear controller (see equation (3.4));

$\mu(t)$ – Hamming distance to the morphology $J = I$ minimized across the Pareto front observed at generation t of the evolution;

$E(t)$ – practical pointing error (see equation (3.7)) minimized across the population at generation t.

3.3 Introduction

To behave adaptively is to respond to changes in the environment in such a way as to achieve a certain goal. A system capable of such a behavior can be seen as an automatic problem solver for a certain range of tasks. A consequence of that is the possibility for natural adaptation mechanisms such as evolution and learning be modeled with problem solvers such as evolutionary or machine learning algorithms [90, 78]. The opposite is also true: when designing systems capable of adaptive behavior, general approaches to problem solving are valuable tools that can be used either directly as a component of the solution [15] or to design a solution for a particular environment automatically [40].

One idea often instantiated in such approaches is to divide the whole problem into subproblems and solve each one of these with some degree of independence from

others. This idea, which we will call the **division approach** * has proven to be effective for a wide variety of tasks. In politics and warfare, the principle *divide et impera* is used at least since antiquity [114]. The approach plays a fundamental role in engineering [3, 115], including algorithmic design [58] and artificial adaptive systems [1].

Whenever the division approach is used, the resulting solution possesses a system property called modularity. It is broadly defined as a capacity of a system consisting of many components to be decomposed into groups of components (modules) such that within each group the dependencies between the components are strong, while the dependencies between the components that are in different groups are comparatively weak. Although the definitions of "components" and "dependencies" vary depending on the nature of the system, all systems that are produced with the division approach are modular at least in one way. In particular, components of the resulting system that are produced by solving a single subproblem all arise from the same solution process and in this sense are more dependent on each other than the components pertaining to different subproblems.

A peculiar property of modularity is that if the process of solving the task is incremental and the modularity is present in a partial solution then even the simplest search methods can provide the same benefits as the division approach. Consider the problem of designing an internal combustion engine with a maximum total energy output. If the optimization process begins with an engine in which all subsystems

*The term was chosen to distinguish from divide-and-conquer methodology in algorithmic design. Division approach is a broader term that encompasses all useful ways to split the problem into subproblems, while divide-and-conquer is its instance that focuses on non-overlapping subproblems. Approaches that involve overlapping subproblems, such as dynamic programming, are also instances of the division approach.

(cooling, lubrication etc.) are tightly coupled, then any chance improvement of one subsystem will likely cause many other subsystems to change their behavior, increasing the probability that the overall change will be detrimental to the performance. On the other hand, if the subsystems are functioning with relative independence from each other, then a possible improvement within one subsystem will not affect other subsystems much, resulting in an overall increase in performance [115]. In incremental setting, this is how the division approach operates: after dividing the problem into subproblems, each of these is solved, to some extent, independently. This pattern can be enforced by the modular structure of the partial solution, or it can be built into the solution process by the designer, resulting in modular partial solutions. Thus, in engineering, whenever the incremental algorithms are used the causal relationship between the solution modularity and the division approach is bidirectional.

In nature, modularity is observed in all biological systems from the molecular to the ecosystem level [19, 45, 122]. Its emergence in biological systems is hypothesized to be related to the reduction in cost of complexity that is associated with it [126]. Several mechanisms by which such emergence may occur have been proposed [76, 107, 56, 122, 31, 36, 26]. Many of these were investigated by evolving model problem solvers, such as genetic regulatory networks or neural networks, with genetic algorithms, and in almost all of these experiments an increased rate of adaptation (i.e. rate of error reduction or fitness increase) was reported to coincide with the emergence of modular solution candidates [76, 56, 36, 26]. Similar improvement was observed even when modularity was introduced into the systems artificially by biasing search towards more modular solutions [33, 7] (*note - [7] is also provided as Section A.1*).

This coincidence appears analogous to the relationship between modularity and

the division approach in the engineered systems. In many models it was indeed observed that evolution produced modular solutions consisting of quasi-independent modules solving subtasks and/or capable of evolving separately [56, 36, 26, 34, 18]. This is a non-trivial observation because it has also been discovered that more modular networks, on average, tend to have smaller number of connections than their less modular counterparts [76, 26, 7] (*note - [7] is also provided as Section A.1*). Thus, the increase of rate of adaptation could be attributed to the decrease in the effective size of the search space, as opposed to the independent optimization of modules.

There is evidence, however, that the viability of modular solutions, together with the benefits they bring, is heavily dependent on the task. In our previous work we provide an example of a task and a robot morphology for which any controller that solves the task must be nonmodular, and we show that the rate of evolutionary adaptation for this task/morphology pair is much slower than the rate of evolutionary adaptation for the same task, but with a morphology that enables modular control [5] (*note - also provided as Chapter 2*). Indirectly, the dependence of the feasibility of modular control on the morphology (and therefore, on the structure of the control task) is hinted upon by the results showing that modular controllers are more likely to evolve if the morphology evolves alongside the control, with the additional objective of behavioral conservatism [14].

3.4 Reformulation

In most of the work cited above, the task itself is considered to be fixed. In engineering, however, this is rarely the case. For almost every real world task there is a

space of possible ways to approach and formalize it. Changing the approach can trivialize tasks that initially appear intractable. Human problem solvers are well known to be able to exploit that dependence. We will call the corresponding cognitive technique – reformulating the problem to make it easier to solve, instead of attacking it directly – **reformulation** (e.g. [25]). In human cognition, reformulation is arguably the instrument of choice when dealing with the hardest tasks. It is widely hypothesized that it underlies many aspects of cognitive insight [32, 127, 113], including the so-called Eureka effect [59, 113].

We formally define reformulation as follows (Figure 3.1A). Consider a finite-dimensional optimization problem, for which solutions are encoded as vectors of N values that minimize some real-valued function. Suppose there is some measure of difficulty for finding such a solution. Examples of such measures include a binary value that indicates whether a certain solution technique worked, or the number of operations required to achieve an acceptable result. Suppose further that changing some subset of M ($M < N$) variables (which we will hereafter call **driving variables**) can significantly change the difficulty of optimizing the $N - M$ remaining **non-driving variables**. The approach is then to isolate the driving variables and optimize their values (i.e., the **formulation** of the problem of optimizing the non-driving variables) to ease of optimization of the non-driving variables.

To instantiate the reformulation approach, a designer must isolate the driving variables, select the two optimizers – for driving and for non-driving variables – and provide an appropriate quantitative definition of "**optimization difficulty**" of the non-driving variables optimization given some values of driving variables. The driving variables must influence the difficulty of the optimization of the non-driving variables

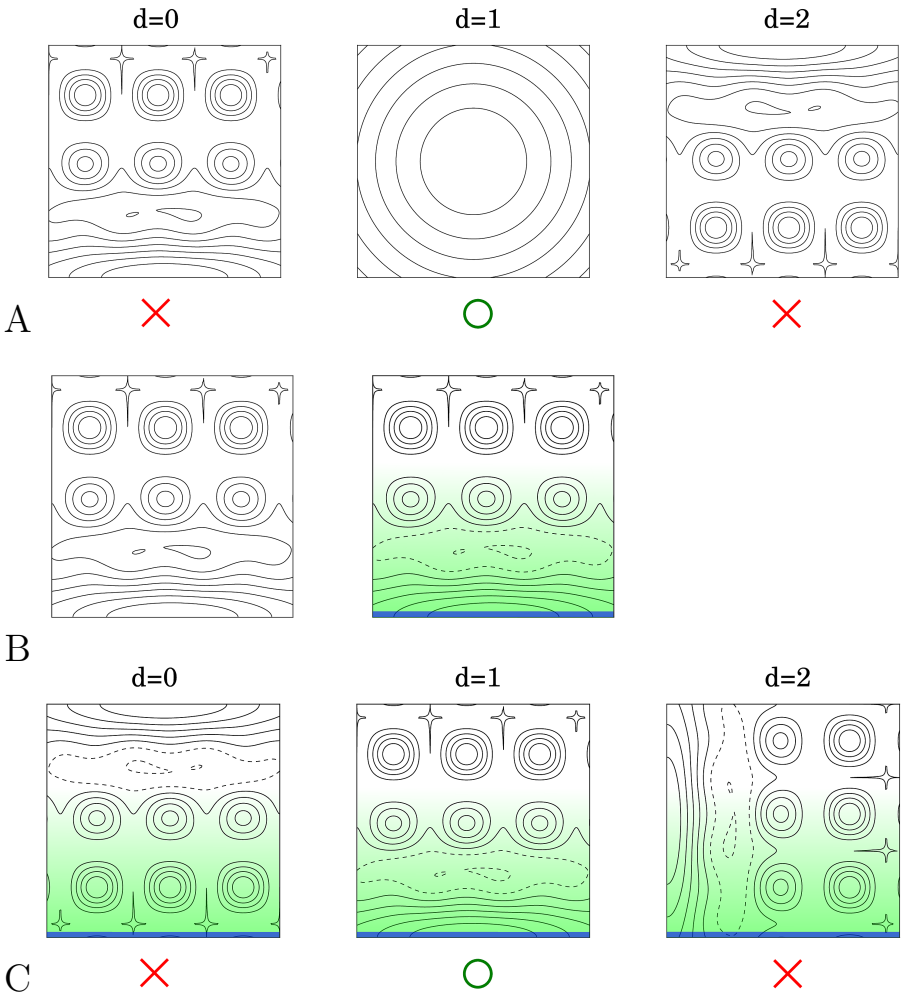


Figure 3.1: Graphical representations of the core approaches of the paper. ***(Row A) reformulation:*** *shape of the goal function landscape depends on some variables (exemplified by d). If for some values of these variables the optimization on the landscape can be done more easily, we call them driving variables. To reformulate the optimization task is to optimize those variables in order to find the values that simplify the underlying optimization process. In our example such a value is* $d = 1$*, corresponding to convex optimization.* ***(Row B) bias:*** *the search is biased (as indicated by the green gradient) towards a subset of the search space (indicated by the blue stripe in the bottom of the square). The technique introduces some assumptions about the fitness, in our example that the fitness landscape is more convex and contains good enough solutions near the blue stripe. If these assumptions are not satisfied, the bias may be detrimental.* ***(Row C) guided reformulation:*** *shape of the goal function landscape depends on some variables (again exemplified by d). However, the difficulty of the optimization does not depend on those variables unless the bias is applied, in which case the variable affects whether the assumptions of the bias are satisfied and thus the optimization difficulty. If the bias is applied, these variables become driving and can be optimized to reformulate the original optimization problem while utilizing the bias. In our example the optimal value is again* $d = 1$*: for that value, the fitness landscape is convex and has good-enough solutions around the area towards which the search is biased.*

with the optimizer that has been selected for these. Selecting such driving variables is a generally a matter of domain knowledge, but some guidelines can be provided (see Section "Guided reformulation"). The definition of difficulty must be **correct** in a sense that it must not assign low difficulty to the driving variables values that prohibit the discovery of good-enough solutions in the optimization of non-driving variables. If this condition is not satisfied, a possibility of premature convergence arises. Additionally, the definition must be significantly less computationally expensive than finding a good enough solution for most values of driving variables, because otherwise it is less computationally expensive to solve the optimization problem without reformulating it.

If both driving and non-driving variables are optimized with incremental algorithms, one natural way of estimating the difficulty is to set values of the driving variables, run the algorithm for the non-driving variables for some number of iterations, and estimate the difficulty based on how much did the fitness improve. We will refer to this difficulty definition as **improvement-based**. The downside of this approach is that its correctness depends on the properties of the fitness landscape: if, for some values of driving variables the fitness improves rapidly, yet its ultimate value is not good enough, then premature convergence is possible.

The number of iterations that are used to estimate the optimization difficulty for every value of the driving variables governs the reliability of this difficulty definition: if this number is unlimited, the definition is correct; if it is small, it is not correct on many fitness landscapes. We will refer to such definitions of difficulty as **approximately correct**. The conflict between the need to spend the iterations on refining the estimates made with this definition and on optimizing the driving variables results

in two timescales: a slow one for the driving variables and a fast timescale for the non-driving variables. This is an instance of separation of timescales of adaptation [92]: a widespread property of natural and artificial systems originally discovered in evolutionary biology [54, 117, 77] and later in neuroscience [38, 124, 119] and machine learning [49, 129, 57].

3.4.1 Evolution and reformulation

Evolutionary computation is an incremental optimization technique that relies on trial and error. It is possible to use evolution to optimize both driving and non-driving variables (e.g. [21]), or to only evolve the driving variables, while the non-driving ones are obtained via other techniques such as gradient descent (e.g. [51, 80]). Evolutionary optimization can be improved with the reformulation and related ideas in many ways, and several major ideas of the field are, in the opinion of authors, related to the reformulation approach.

Canalization [120, 51] can be described as a phenomenon of discovering a set of values of driving variables that contains much of the information about optimal non-driving variable values discovered with the corresponding optimizer. It occurs when the design variables are divided in two subsets, one in which the variable values are learned during the lifetime and the other, in which they are learned by the evolution. The variables under the evolutionary control affect the difficulty of lifetime learning and thus can be thought of as driving variables, with each set of values being a "learner". Learner's fitness is defined as a nondecreasing function of the proportion of the learner's lifetime that it spends with all the variables having exactly the right values; such fitness is a correctly defined difficulty of optimizing the non-driving

variables. Lifetime learning of the non-driving variables enables the evolution to evaluate a mutation-continuous subset of the variable values rather than a single point, effectively smoothing the fitness landscape. As long as the values are correct, the more information on the non-driving variable values is contained within the driving variable values, the easier the lifetime learning is; hence, there is an evolutionary pressure to store all the information on the non-driving variable values within the driving variables. Thus, the characteristics learned over lifetime (non-driving variable values) are assimilated into the genome (driving variable values) over the evolutionary time, ultimately producing a formulation that contains all the information on the solution.

A development of the idea of canalization is evolution for evolvability [60, 125], where variables describing genotype-to-phenotype map can bias the evolution of underlying variables towards local optima and improve the rate of convergence when the population is near them. Such genotype-to-phenotype map can be thought of as a vector of driving variables. Evolving it in a slower timescale compared to the rest of variables and interleaving the periods of neutral and adaptive evolution forces the evolution to generalize over multiple ways of reformulating the underlying problem.

Morphological protection [21] can be seen as a method for circumventing the incorrectness of the improvement-based difficulty definition. It investigates the evolution of morphology alongside the controller in robotics control tasks. Morphology governs the complexity of control and thus is a driving variable. Protecting values of this variable (i.e., formulations of the control problem) from modification for the period while the control is optimized enables these values to compete based on the final fitness of controllers that can be evolved for them and not the fitness improvement over any fixed number of generations.

In co-evolution [2, 87, 98] two groups of variables are optimized to make each other's optimization more difficult. Thus, co-evolution can be thought of as an approach similar to reformulation, but with positive influence of one optimization process on another's rate of convergence replaced with a negative feedback. While evolutionary reformulation is good at producing simple solutions, co-evolution tends to produce the solutions that complexify over evolutionary time, which can be valuable in many applications.

Aside from fairly general approaches, some evolutionary algorithms for more narrow classes of tasks can also be seen as instances of the reformulation approach. One example is GPESA, a method that evolves a pattern of aggregation of geospatial variables alongside a genetic programming model for value prediction [63]. Another is the evolution of deep learning networks [80], in which the topology of the network is evolved to maximize its learning rate.

A similar but different procedure to the reformulation is optimization of hyperparameters in machine learning (e.g. [9]). The difference is in the designer's intention: in machine learning hyperparameter optimization is typically done in order to improve the final model, while reformulation aims at simplifying the *process* of constructing the model, in order to find good-enough solutions to the toughest tasks.

3.4.2 Guided reformulation and its application to evolutionary robotics

One difficulty in using the reformulation approach is the need to select the driving variables. In human cognition, the method is typically "thinking outside the box":

finding some controllable variables that influence the difficulty of the task, but of which the designer is initially unaware. However, whether a given variable does or does not influence the difficulty of the task depends on the method that is used to optimize the non-driving variables. This opens up a possibility for either making certain variables into driving ones or increasing the influence that some known driving variables exert on the solution difficulty. Here we will explore an approach that uses that possibility.

One powerful approach to improving the performance of the optimization is to bias the search (Figure 3.1B). The bias prioritizes the solution candidates from a subset of the search space in which, based on the domain knowledge, some reasonably good solutions can be expected, or which has a reasonably good probability of intersecting a hill climbing path towards such solutions. For example, if an embodied agent such as human or animal searches the area that can only be spotted from a short distance, a common procedure is to follow a linear trajectory that covers the area in such a way that the distance from every point of the area to the trajectory is reasonably small. The trajectory is also typically organized in such a way that the areas where, according to the prior knowledge, the object can be found with higher probability are searched earlier. In machine learning, the same idea can be implemented with indirect encodings [110] or by co-optimizing some heuristic parameter together with the goal function. Some examples of such parameters relevant to neural network optimization are connection cost [26] and L2 norm of the weights [9] (both of these should be minimized for a useful bias).

This approach effectively reduces the size of the search space and minimizes, based on the prior knowledge, the average time to arrive to the solution. It can also be used

to select a solution with useful properties if multiple good-enough solutions exist. However, it does so at the cost of introducing some assumptions about the solution and the search space. If the assumptions do not hold, the bias can be detrimental. For example, in our previous work [5] (*note - also provided as Chapter 2*) we've shown that for certain control tasks strengthening the bias towards sparsity can make the evolutionary optimization much more difficult.

Suppose that, for some bias, some subset $\mathcal{A}$ of design variables can influence whether the assumptions introduced by the bias hold or not, with respect to the optimization of the remaining variables. If the assumptions introduced by the bias hold, the usage of the bias is likely to have a drastic impact on the difficulty of the problem of optimizing the variables not in $\mathcal{A}$. Thus, as long as the bias is used in the optimization of the variables not in $\mathcal{A}$, the variables in $\mathcal{A}$ are *driving variables*. They then can be used to reformulate the problem in such a way that the assumptions introduced by the bias do hold and the usage of the bias improves the performance of the optimization.

We will call the resulting approach **guided reformulation** (Figure 3.1C). It can be summarized as follows:

> To reformulate an optimization problem, find a pair of an optimization bias and a set of variables such that the assumptions introduced into the optimization by the bias are dependent on the variables in the set. The reformulation procedure that utilizes the variables in the set as driving and uses the bias in the optimization of the non-driving variables is then likely to benefit from the bias (typically by producing good-enough solutions selected for additional properties by the bias more rapidly than in absence

of bias or reformulation procedure).

Note that this approach does not aim to eliminate the need for domain knowledge, but to provide some guidance on how to use it instead. Fully automatic instantiation of the reformulation approach is outside of the scope of this paper.

One particularly powerful type of bias that can be utilized in guided reformulation is the bias towards modularity (see Introduction). Resistance to catastrophic forgetting and (in network optimization) correlation with sparsity can make evolutionary algorithms biased towards modularity converge much more rapidly than their counterparts with no such bias [26, 7] (*note - [7] is also provided as Section A.1*). Here we describe a way to use guided reformulation to get the benefits of the bias towards modularity in a robotic control task, building on the body of research outlined in Introduction.

We consider a task of controlling a robotic agent embedded in physical space. From our previous work [5] (*note - also provided as Chapter 2*) we know that the capacity to admit modular control can depend on an agent's morphology. The results from [5] (*note - also provided as Chapter 2*) also suggest that such capacity is sufficient for the bias towards modularity to make the convergence much more rapid than in the absence of the bias. We will refer to this statement as **hypothesis 0**. If this hypothesis is correct, then whenever the morphological variables influence the capacity of the agent to admit modular control they also qualify as driving variables under the bias of control optimization towards modularity.

Due to the generality of hypothesis 0 it is impossible to confirm experimentally. Instead of attempting to do that, we adopt falsificationist attitude towards it. To this end, we formulate some of its consequences and attempt to show that they are false.

We instantiate guided reformulation by optimizing morphology alongside the controller, parameters of which are the non-driving variables in this case. We adopt a genetic algorithm as the optimization technique for both sets of variables and separate the optimization of driving and non-driving variables by using different mutation rates for morphology and control. If the mutation rate for control is greater than for morphology, many controllers are considered for every morphology, ensuring that the current values of error of evolutionary individuals (composed of a morphology and a controller) provide approximate definitions of difficulty of optimization of control. The bias towards modularity in control optimization is implemented in two ways: using the connection cost technique [26] and with initial populations of sparse networks [7] (*note - also provided as Section A.1*). It can also be switched off.

The approach is applied to the "Arrowbot" task and the environment from [5] (*note - also provided as Chapter 2*).

In this setup, two premises additional to hypothesis 0 are important:

Premise 1: it is known that for the "Arrowbot" task and environment the capacity for modular control depends on certain morphological variables. This is established in [5] (*note - also provided as Chapter 2*).

Premise 2: evolutionary algorithm that we employ is capable of following error gradients and converge on error minima (i.e. it is a functioning search algorithm). This is ensured by elitism of the algorithm: at every change of generation the best performing individual is preserved.

We predict the following:

Hypothesis 1: within each run, rapid convergence of the control evolution will fol-

low the discovery of a morphology that admits modular control. Follows from hypothesis 0 and premise 1.

Hypothesis 2: evolution of morphology will converge on body plans that admit modular control. Premises ensure that there is a gradient of control optimization difficulty for the morphological evolution to follow, and that the minima of such difficulty will coincide with body plans that admit modular control. Follows from hypothesis 0, premise 1 and premise 2.

Hypothesis 3: evolving the control with a randomly selected fixed morphology will result in a less rapid convergence than evolving the morphology alongside the control. Equivalently, the convergence rate will be lower if the morphological evolution cannot follow its gradient. Follows from hypothesis 0, premise 1 and premise 2.

Hypothesis 4: without the bias towards modularity, convergence to body plans that admit modular control will likely not happen or will happen in a larger number number of generations than in the case when the bias is enabled. From hypothesis 0 and premise 1 we conclude that the impact of morphology on the rate of convergence of control evolution is dependent on the presence of the bias. With the bias switched off, morphology's impact on the rate of convergence of control evolution will be decreased, causing the gradient guiding the morphological evolution to disappear or weaken.

Hypothesis 5: by the same reasoning, reducing or disabling the bias towards modularity will cause a decrease in the adaptation rate.

Hypothesis 6: as long as the bias towards modularity is switched on, the qualitative behavior of the system should not depend on the details of implementation of the bias. Equivalently, the status of morphology as a vector driving variable should depend on the presence of the modularity bias, but not on how it is achieved. This follows from hypothesis 0 and premise 1.

Hypothesis 7: there should be a ratio between the mutation rates of morphology and control that maximizes the convergence rate and is not zero or one, but is biased such that more control mutations than morphological mutations occur. This follows from the tension between the need for the difficulty definition to be as correct as possible and the need to advance the optimization of morphology, all within the same iterated optimization process.

The expected behavior of the system integrated from all the hypotheses above is shown in Figure 3.2.

We show the hypotheses 1-5 and 7 to be correct as long as hypothesis 0 is clarified as follows: discovery of a morphology that admits modular control is a necessary but not sufficient condition for subsequent rapid convergence of controller evolution. We fail to obtain any results that contradict hypothesis 0 clarified in that way and hypothesis 6. We investigate how the results behave as the task is scaled up.

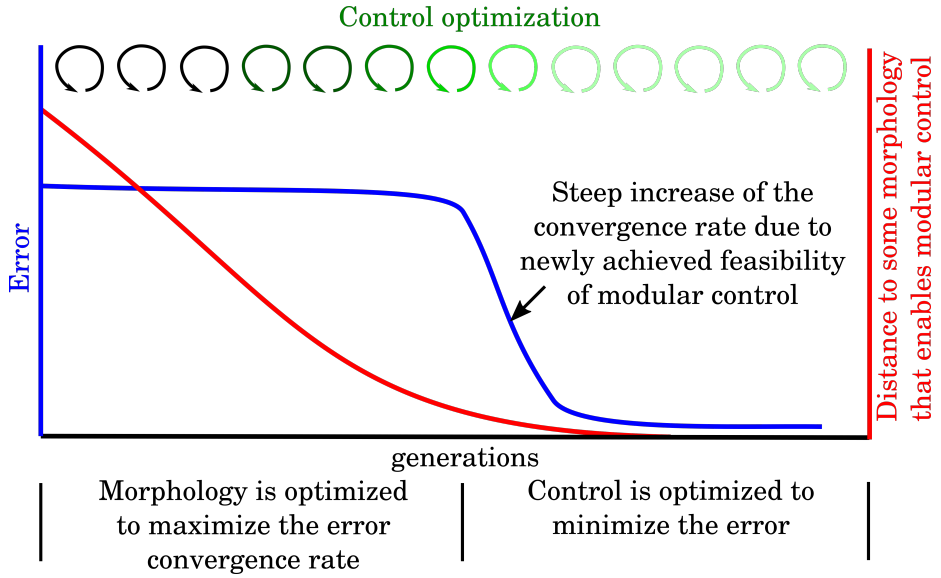


Figure 3.2: Expected behavior of evolutionary guided reformulation approach that uses morphology a vector driving variable and modularity as optimization bias for control (the non-driving variables). To exploit the speedup of the evolution that we hypothesize to arise for morphologies that make modular control feasible, we evolve the morphology alongside the control, but at a slower timescale; additionally, we bias the evolution of control towards modularity. Morphologies that admit modular control enable more rapid reduction of error and thus get an evolutionary advantage. The fast optimization timescale is shown with circular arrows at the top of the graph, with color intensity of the arrow signifying the difficulty of control optimization at that point of evolutionary history. If the task permits controllers consisting of a multitude of modules for some morphology, the error reduction rate will radically increase as this morphology is approached. Note that what the morphology influences is the rate of convergence, not the error itself.

3.5 Methods

In this section we describe in details the task, environment and the family of robotic morphologies that we used to test our hypotheses and also the genetic algorithm we used.

3.5.1 Robot

We here study the interaction between morphological evolution and modularity-biased controller evolution using Arrowbots, a robotic substrate we previously used to demonstrate the influence of morphology on the feasibility of modular control and its evolution [5] (*note - also provided as Chapter 2*).

An Arrowbot is a robot which consists of N segments with pointers stacked to form a column (fig. 3.3). Adjacent segments are connected with motors that share a common geometric axis of rotation, that is, the vertical axis; the bottommost segment is connected to a fixed base. Any motor input controls the angular velocity of the segment that it connects to, relative to the segment below it, or, in the case of the motor connected to the first segment, the fixed base. We will denote those relative orientations as r_i [†] , the angular velocities as $\dot{r_i}$ and the corresponding motor inputs as m_i.

Absolute orientations of the segments will be measured with respect to the fixed

[†]Note that in our treatment, all orientations do not wrap around, i.e. angles ϕ radians and $\phi + 2k\pi$ radians, $k \in \mathbb{Z}$ are treated as *different*. In this treatment there are no special angle values and the need to keep track of angle units is eliminated.

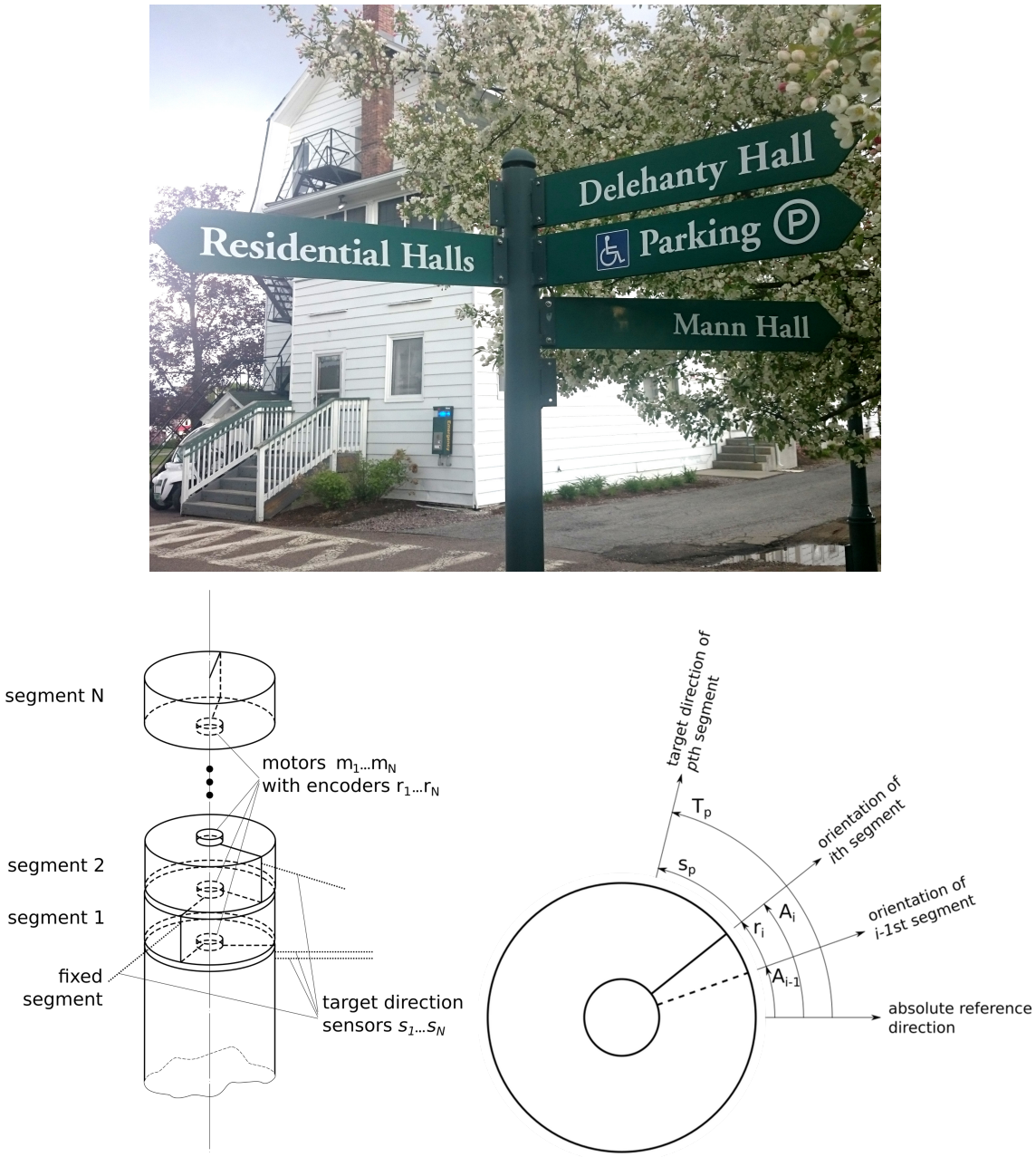


Figure 3.3: Multi-directional road sign ***(top)****, its dynamic version, the Arrowbot* ***(middle)*** *and the associated kinematic notation* ***(bottom)****.*

base and denoted as A_i. The relationship between the two is as follows:

$$\begin{aligned} A_i &= \sum_{k=1}^{i} r_k \\ r_i &= \begin{cases} A_1 & \text{for } i = 1, \\ A_i - A_{i-1} & \text{for } i = 2..N \end{cases} \end{aligned} \tag{3.1}$$

For convenience, later we will also use the vector notation:

$$\begin{aligned} \boldsymbol{A} &= K\boldsymbol{r} \\ \boldsymbol{r} &= K^{-1}\boldsymbol{A} \\ K &\equiv \begin{bmatrix} 1 & 0 & ... & 0 \\ 1 & 1 & ... & 0 \\ & . & . & \\ 1 & 1 & ... & 1 \end{bmatrix} \\ K^{-1} &= \begin{bmatrix} 1 & 0 & 0 & ... & 0 & 0 \\ -1 & 1 & 0 & ... & 0 & 0 \\ 0 & -1 & 1 & ... & 0 & 0 \\ & & . & . & . & . \\ 0 & 0 & 0 & ... & -1 & 1 \end{bmatrix} \end{aligned} \tag{3.2}$$

Arrowbots are equipped with sensors of two kinds: N proprioceptive sensors that directly measure the orientations of the segments they are attached to $\boldsymbol{r}$ and N target orientation sensors $\mathbf{s}$. Any target orientation sensor s_i outputs the difference between the orientation of the object it is attached to and the target orientation T_i. Possible attachment points for any sensor include any segment as well as the fixed base. For

example, the sensor s_3 always measures the difference between the target direction of the third segment T_3 and whatever it is attached to. If it is attached to the second segment, it will output $T_3 - A_2$; if it is attached to the fixed base, it will output just T_3.

More generally,

$$\mathbf{s} \equiv \boldsymbol{T} - J\boldsymbol{A}, \tag{3.3}$$

where J is an $N \times N$ matrix such that $J_{ij} = 1$ if the jth target orientation sensor s_j is attached to the ith segment and 0 otherwise. Within a fixed environment, the matrix J fully determines the effect that any motor action of an Arrowbot has on its sensors; therefore, it determines its morphology. Hereafter we will use the terms "morphology", "target orientation sensor attachment pattern" and "the matrix J" interchangeably.

A controller of an Arrowbot is the part that takes the sensor readings as inputs and feeds its outputs to the motors. It can be abstracted as a mapping $\mathbf{f} = \mathbf{f}(\mathbf{s}, \boldsymbol{r})$. Throughout this paper we will discuss the general case of nonlinear control and its properties; for the experimental part of the paper, however, we will use linear control:

$$\mathbf{f}(\mathbf{s}, \boldsymbol{r}) = W\mathbf{s} + Y\boldsymbol{r}. \tag{3.4}$$

Further, we limit the elements of the matrices W and Y to be in the set $\{-1, 0, 1\}$ to reduce the size of the search space.

To define the connectivity of such controllers we represent them as graphs. Each sensor or motor is represented as a node and each nonzero coefficient in the matrices W and Y is represented as a bidirectional connection between a sensor node and a

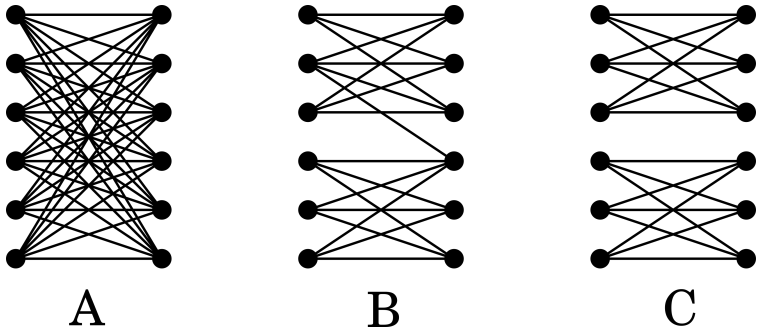


Figure 3.4: Types of network modularity in linear controllers. ***(A)*** *Fully connected controller is nonmodular.* ***(B)*** *A controller with high Q, but with a single connected component.* ***(C)*** *A controller with two connected components. To use analytical results from (Bernatskiy & Bongard, 2017) we must require complete independence between modules within the controller for the controller to be called modular. Hence, only the networks of type (C) are considered modular within the present paper.*

motor node. With this definition, the analysis of the dependence (defined as in [5] (*note - also provided as Chapter 2*)) is reduced to the analysis of connectivity of the graph it is represented with: whenever there is a connected path between any two nodes, they are dependent. Thus, the connected components of the controller graph represent groups of variables in which every variable is dependent on every other one.

We will consider a controller to be modular if its graph has more than one connected component. This definition will be used for consistency with [5] (*note - also provided as Chapter 2*), which is required for using some analytical results from that work. Compared to the more traditional Q metric [85], this definition provides a more coarse grained picture in which modular networks are guaranteed to have at least two groups of completely independent variables (see Figure 3.4). This, in turn, guarantees that the connections within each group can be changed without influencing any other group through the controller, ensuring that the pattern of adaptation characteristic of the division approach (see Introduction) is induced within any trial-and-error

optimization method with localized changes in connections.

Knowing the morphology and the controller, it is possible to determine the dynamics of an Arrowbot:

$$\dot{\boldsymbol{r}} = \mathbf{f}(\boldsymbol{r}, \mathbf{s}(\boldsymbol{r}, \boldsymbol{T})), \tag{3.5}$$

where $\mathbf{s}(\boldsymbol{r}, \boldsymbol{T})$ is given by equation (3.3).

3.5.2 Task

The task of the robot is to orient its pointers in arbitrary target orientations $\boldsymbol{T}$ starting the movement at arbitrary initial conditions r_0. Ideally, the performance of the robot should be measured with all possible settings of initial conditions and target orientations; additionally, since we're only interested in the final pointing error and not the transient time, the evaluation times should be infinite. The performance in this case can be quantified as

$$E_{ideal} = \sup_{\{\boldsymbol{T}\},\{\boldsymbol{r}_0\}} \lim_{t\to\infty} \left(\boldsymbol{T} - \boldsymbol{A}(t)|_{\boldsymbol{r}_0,\boldsymbol{T}}\right)^2 \tag{3.6}$$

where $\mathbf{x}^2$ is a scalar product, $\mathbf{x}^2 \equiv \sum_i x_i^2$, and $\{\boldsymbol{T}\}$ and $\{\boldsymbol{r}_0\}$ are sets of all possible target orientation vectors and initial conditions vectors, correspondingly.

In practice we evaluated the performance of the robot by averaging its pointing error after a finite simulation time t over a finite sets of initial conditions and target orientations:

$$E = \frac{1}{n_{IC} n_{TO}} \sum_{\alpha=1}^{n_{IC}} \sum_{\beta=1}^{n_{TO}} \left(\boldsymbol{T}^\beta - \boldsymbol{A}(t)|_{\boldsymbol{r}_0^\alpha,\boldsymbol{T}^\beta}\right)^2, \tag{3.7}$$

where n_{IC} and n_{TO} are the numbers of variants of initial conditions $\boldsymbol{r}_0^\alpha$ and target

orientations $\boldsymbol{T}^{\beta}$ that the robot is being tested at. In particular, throughout this paper we used the following set of N target orientation vectors:

$$\boldsymbol{T}^1 = [1, 0, ..., 0], \boldsymbol{T}^2 = [0, 1, ..., 0], ..., \boldsymbol{T}^N = [0, 0, ..., 1]. \tag{3.8}$$

Each of these target orientation vectors was tested with just one set of initial conditions $\boldsymbol{r}_0 = [0, 0, ..., 0]$. Thus, $n_{IC} = n_{TO} = N$. $\boldsymbol{A}(t)$ is obtained by integrating equation (3.5) forward in time over the span of t with the fourth order Runge-Kutta method. Fitness is is computed using equation 3.7 using the final state of the system.

The system (3.5) exhibits linear dynamics, resulting in a tendency for an exponential growth or decay of r over time. This results in heavy tailed distributions of the derived variables such as the approximate error E for Arrowbots with randomly generated morphologies and/or controllers. To simplify the statistical analysis of our data, we account for this tendency by running all statistical tests on the decimal logarithm of error, $\log_{10} E$.

3.5.3 Genetic encoding and mutation

In this paper, we encoded each Arrowbot as a concatenation of two vectors of integers.

The first vector encodes the morphology of the Arrowbot in N integers from $\{0, 1, ..., N\}$. An integer at the ith position is the number of the segment to which the ith target orientation sensor s_i is attached, with the value of 0 representing attachment to the fixed base. It is thus a compressed encoding of the sparse matrix J (equation (3.3)).

The second part encodes the linear controller (equation (3.4)) as $2N^2$ integers drawn from $\{-1, 0, 1\}$. The first N^2 elements of the genome are the elements of the matrix W and the second half are the elements of Y, reshaped to form a linear array.

Random genomes for initial populations were generated by concatenating a vector of N integers randomly selected from $\{0, 1, ..., N\}$ (the morphology) with an encoding of a randomly generated controller. We compare two methods of generating the controllers randomly in this paper. The default method for producing the controller is to generate a vector of $2N^2$ integers randomly selected from $\{-1, 0, 1\}$. This produces a linear controller in which about 2/3 of the coefficients are nonzero; we will refer to such controllers and to initial populations composed of them as **dense**. The alternative is to use the method introduced in [7] (*note - also provided as Section A.1*): take a vector of $2N^2$ zeros and mutate it once using the control mutation operator described below. Due to the structure of the operator, such controllers will have exactly one nonzero coefficient drawn from $\{-1, 1\}$ and placed randomly. We will call such controllers and initial populations composed of them **sparse**.

We use a mutation operator that always changes the genome upon application. With a fixed probability P_{mm} it changes the morphology; failing that, the controller is changed.

We considered two ways of performing a morphological mutation. By default, a randomly selected sensor was moved by one segment up or down, unless the sensor ended up below the fixed base or above the Nth segment as a result; in these cases, the operation was repeated starting with selecting the sensor randomly. The alternative was to discard the morphology altogether and replace it with a newly generated one. We will refer to this latter method as a random jump in the morphological space.

For control mutations, we employed the same operator as in our previous work ([5, 7] (*note - also provided as Chapter 2 and Section A.1, correspondingly*)). It treats the controller as a graph as described in Section "Robot and task". Each field of the weight vector represents a possible connection. Mutation can result in one of three outcomes: addition of a connection (probability 0.1), deletion (probability 0.1) and a density-preserving change in the network (probability 0.8). If a connection is to be added, a field with the value of zero is found, a value from $\{-1, 1\}$ is randomly selected, and the field is set to the value. Deletion randomly selects a field with nonzero value and sets it to zero. Density-preserving mutation flips the sign of the value at a randomly selected nonzero field. If the operation is impossible (e.g. deletion is attempted on a network with no connections), the outcome selection is repeated until a feasible operation is selected.

3.5.4 Evolutionary algorithm

Our evolutionary algorithm is based on the simplified version [7, 5] (*note - also provided as Section A.1 and Chapter 2, correspondingly*) of the biobjective performance and connection cost Pareto optimization technique introduced by [26].

Here, the two objectives of the algorithm are minimization of the pointing error (equation (3.7)) and minimization of the number of nonzero-weight connections in the controller. The algorithm starts by generating an initial population consisting of a fixed number of genomes as described in the previous section. At each generation, a stochastic Pareto front is constructed as follows. Each genome is compared with every other genome in the population. Within each comparison, with probability P_{CC} the first genome is marked as dominated if the second genome has both lower error

and less connections, or, if genomes are equivalent with respect to these objectives, if the first genome was generated or mutated earlier than the second. With probability $1 - P_{CC}$, the first genome is marked as dominated if just its error is greater than the second genome's error. Genomes that are not marked as dominated at the end of the procedure constitute the stochastic Pareto front.

For the values of the constant P_{CC} not in $\{0, 1\}$, there is a non-negligible probability that every genome in the population will be marked as dominated, resulting in an empty stochastic Pareto front. Whenever that happens, a genome with the smallest error is added to the Pareto front.

All of the genomes that are not on the stochastic Pareto front are removed from the population and replaced by offspring of genomes from the stochastic Pareto front. The population size is thus kept constant.

The parameter P_{CC} is added to control the relative emphasis evolution places on the two objectives [26]. Here, we use it to investigate the role of connection cost in the search for morphologies admitting modular control.

3.5.5 Prior knowledge of the morphospace

It has been shown [5] (*note - also provided as Chapter 2*) that for the Arrowbots there are two morphologies with radically different properties with respect of possibility of modular control (Figure 3.5).

If every target orientation sensor is placed on the segment for which it tracks the target, the sensor placement matrix J is equal to the $N \times N$ identity matrix I. In this case every target orientation sensor measures the contribution of the segment it is attached to the total pointing error, $r_i + A_i$. We will refer to this morphology

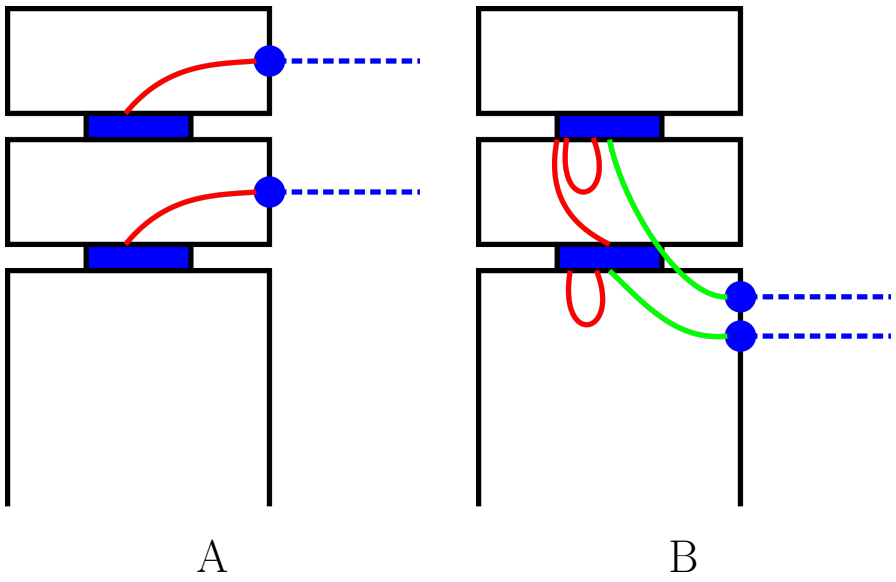


Figure 3.5: Some Arrowbot morphologies with certain known properties for $N = 2$. Blue circles with dotted lines represent target orientation sensors; blue rectangles represent motors with proprioceptive sensors within. The red and green lines show an example of an optimal (with respect to the ideal error (3.6)) controller for each morphology: red for negative and green for positive feedback. ***(A)*** *The $J = I$ morphology that can be controlled by a controller made of N disconnected modules.* ***(B)*** *For the $J = 0$ morphology, provably no controller that is optimal can have more than one connected component.*

as $J = I$ (Figure 3.5A). Previously we've shown that for this morphology there is a family of linear controllers with N disconnected modules with one connection (from s_i to m_i) each that reduces the ideal error (equation (3.6)) to zero.

It is also easy to see that this morphology admits controllers that have the maximum number of disconnected modules. Any graph in which the connections are only possible between the N motor nodes and sensor nodes with $N - 1$ or less connections will necessarily have a disconnected motor node, preventing the pointing error from converging to zero unless the initial orientation of the corresponding segment is aligned with the target orientation. Thus, the task cannot be solved without at least N connections that can form up to N disconnected modules.

Another morphology corresponds to the case when all of the target orientation sensors are attached to the fixed base. In this case all elements of the matrix J are zeros, $J = 0$, and the target orientation sensor measure the absolute angular positions of the targets (Figure 3.5B). For this morphology any controller that reduces the ideal error to zero must be fully connected, as shown in [5] (*note - also provided as Chapter 2*).

In [5] (*note - also provided as Chapter 2*) we found that evolution decreases the final pointing error much more rapidly for the $J = I$ morphology, compared to the morphology $J = 0$, and that the difference becomes more pronounced as the task is scaled up to more segments.

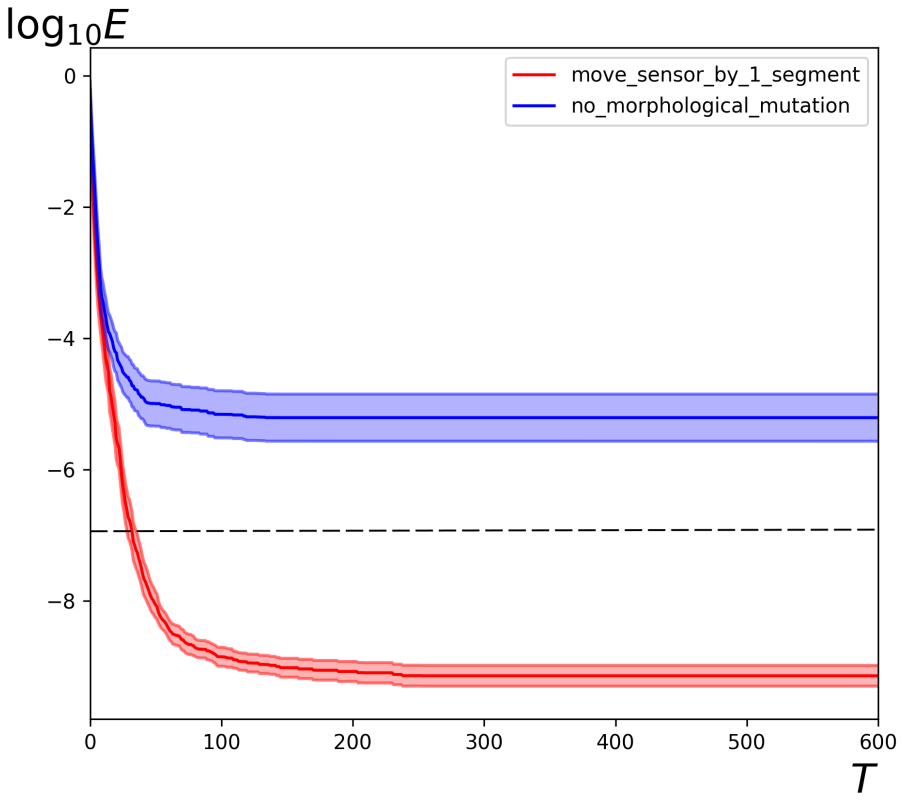
Here we additionally provide a way to design a controller for any morphology that reduces the ideal pointing error (equation (3.6)) to zero and provides a constant baseline performance that does not depend on the morphology for the practical error (equation (3.7)). See Appendix A for details.

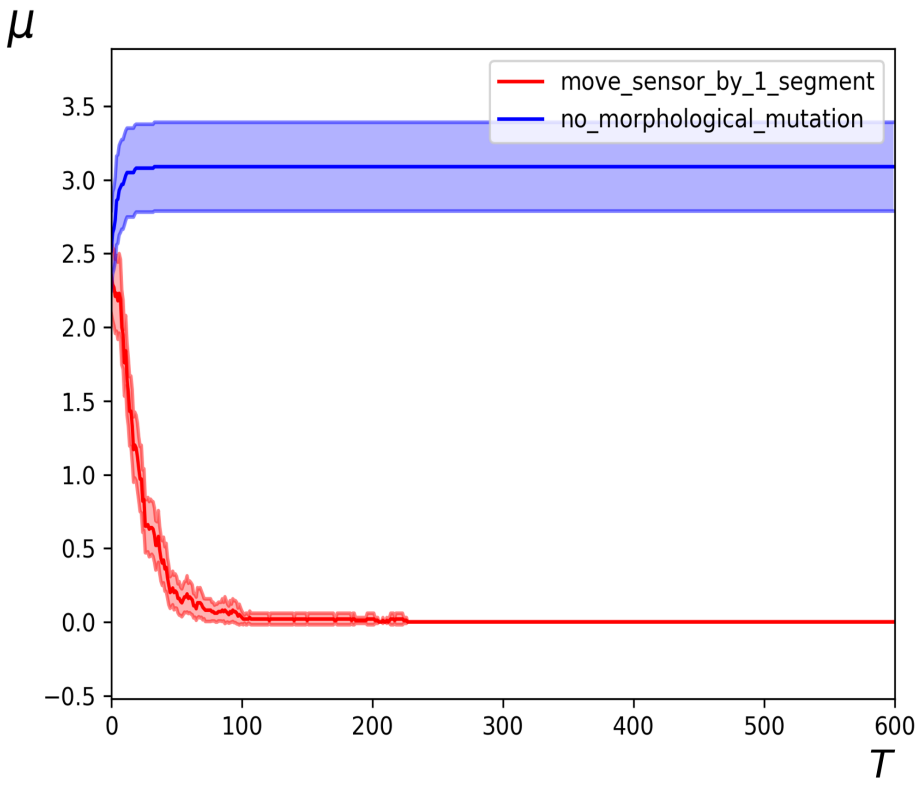
3.6 Results

We began by comparing the performance of the evolution for Arrowbots comprised of three segments with morphological mutation enabled ($P_{mm} = 0.2$) and without morphological mutations ($P_{mm} = 0$) with initial populations of dense controllers. For each on these settings, we performed 100 evolutionary runs and computed statistics on the decimal logarithm of final pointing error (Fig. 3.5, left panel). We have found that for the case when morphological mutations were possible, evolution converged more rapidly and after 600 generations achieved a lower final error, consistent with the hypothesis 3.

Next, we analyzed morphologies that were evolved. We have found that in all of the runs with morphological mutation the $J = I$ morphology was discovered. We verified this using the new parameter μ - the minimal Hamming distance of the morphologies of the genomes on the Pareto front to the $J = I$ morphology (Fig. 3.5, middle panel). As expected, in the absence of morphological mutation the parameter did not change much throughout the run and stayed around a value of three which is typical for initial populations of randomly generated genomes. However, with morphological mutation, the parameter μ reached zero in every run, consistent with the hypothesis 2.

We have also investigated the causal relationship between convergence and the discovery of the $J = I$ morphology. We arbitrarily defined the moment of convergence as the generation t_b when the error goes below the baseline performance of a hand-designed controller (see Appendix A). We define τ_{conv} as the difference between the


$\log_{10}E$
move_sensor_by_1_segment
no_morphological_mutation
0
−2
−4
−6
−8
0
100
200
300
400
500
600
T


μ
move_sensor_by_1_segment
no_morphological_mutation
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
0
100
200
300
400
500
600
T

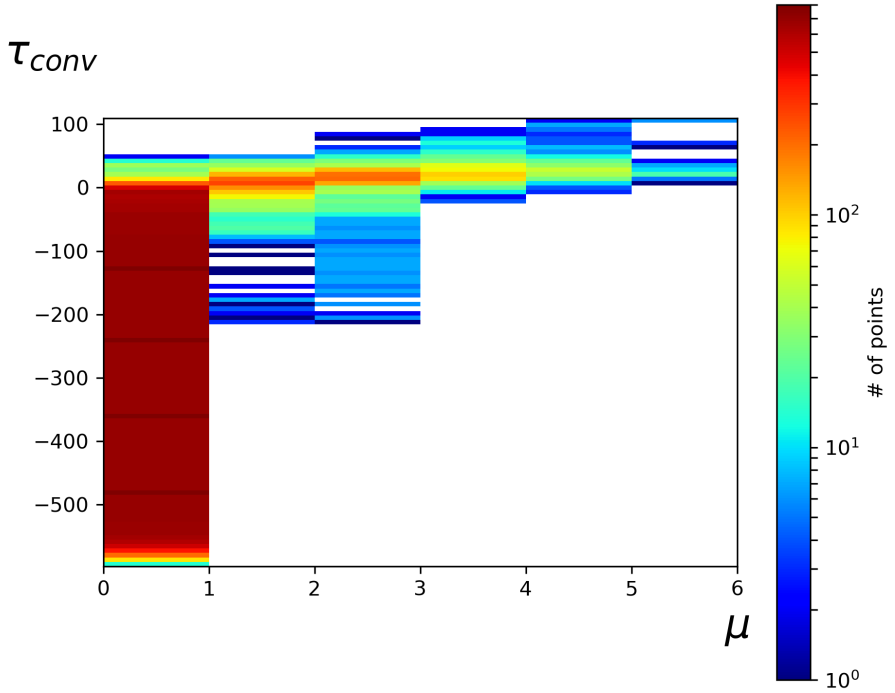


Figure 3.5: Evolving morphology alongside control facilitates the convergence of evolution of the latter and leads to a morphology admitting modular control. ***(top, previous page)*** *Error time series for evolution without morphological mutations ($P_{mm} = 0$, blue line and error strip) and with morphological mutations that move a randomly selected sensor by one segment in random direction ($P_{mm} = 0.2$, red line). The time series were obtained by evolving 3-segment Arrowbots with a population size of 50. Solid lines represent averages and 95% confidence intervals of the decimal logarithm of error based on a sample of 100 evolutionary runs. Dashed line represent the baseline level of performance for $N = 3$ that is achievable for any morphology (see Appendix A).* ***(middle, previous page)*** *Time series of the minimal Hamming distance μ to the morphology $J = I$ across the Pareto front for the same setups. The initial change is due to the evolution utilizing the diversity of the morphologies present within the initial population.* ***(bottom, right above this caption)*** *Defining the moment of convergence as the generation when the error goes below the baseline performance level and the time to convergence τ_{conv} as the difference between the current time and this moment, we consider the state of the population at each generation as a point on the $\mu - \tau_{conv}$ plane. The figure is the density plot for such points. It can be observed that in the majority of runs, convergence occurs after the morphology $J = I$ is found ($\mu = 0$).*

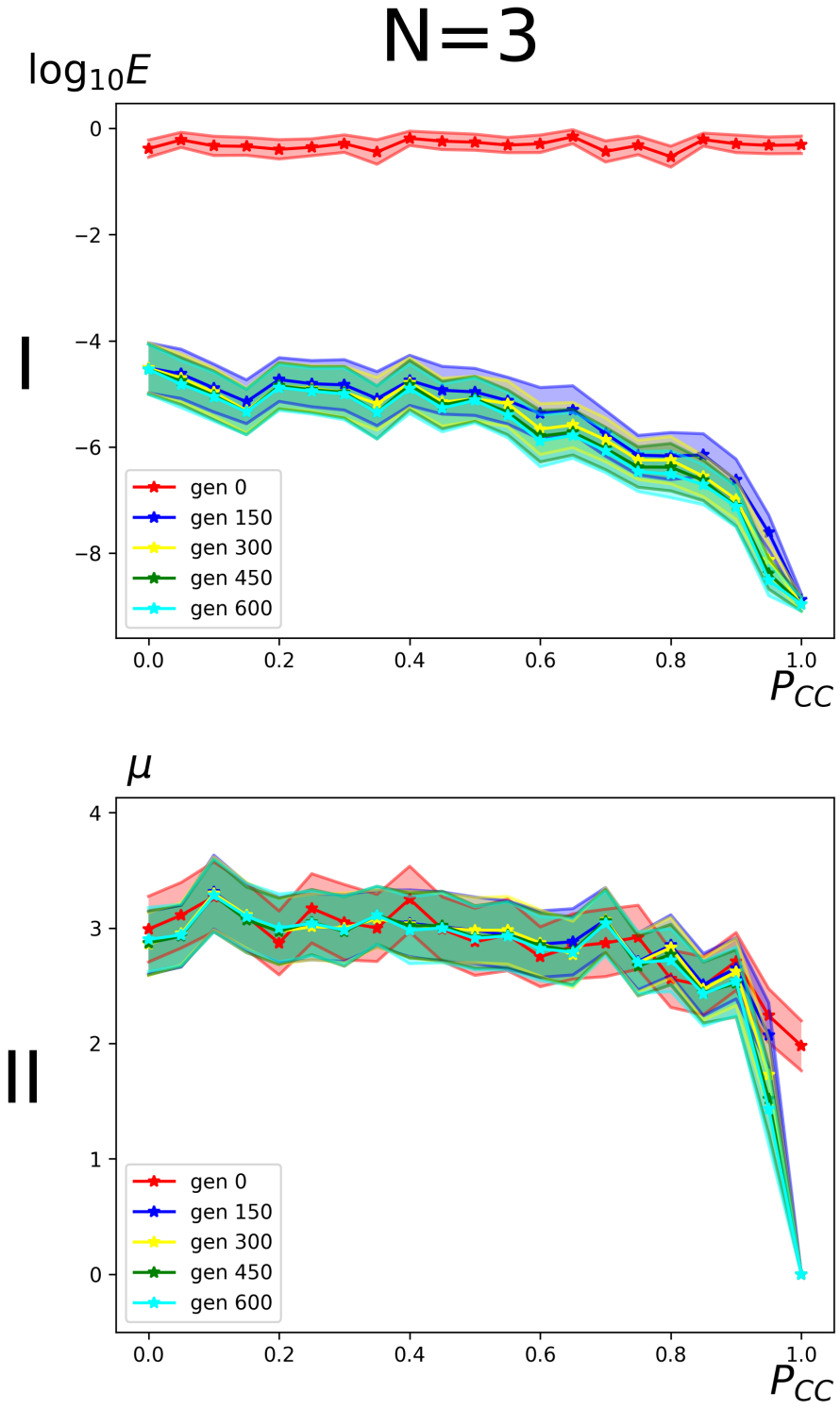
Table 3.1: Parameters of evolutionary runs for experiments involving Arrowbots of different sizes.

Number of segments N	Population size	Total evolutionary time
3	50	600
5	100	1600
10	200	4000

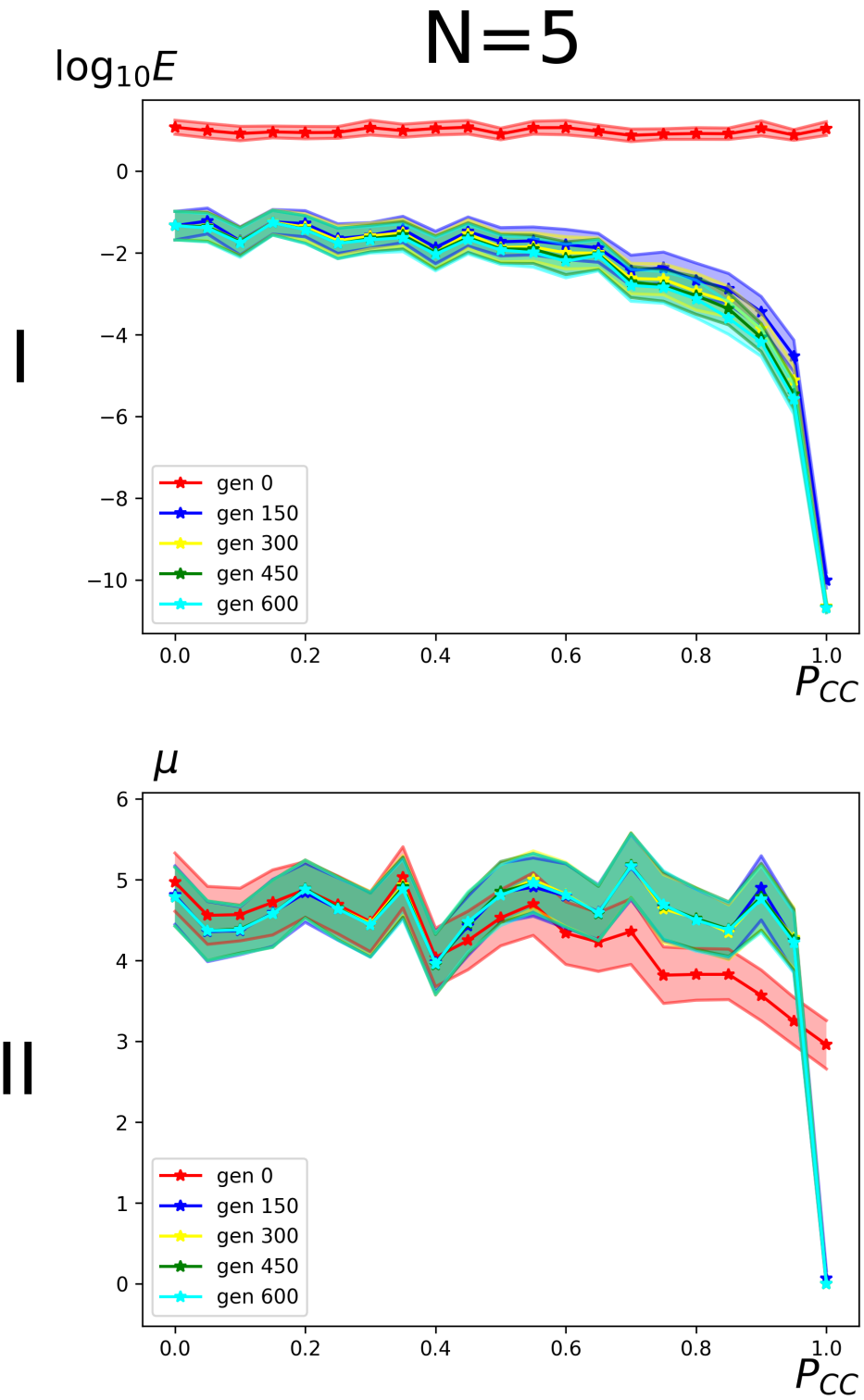
current evolutionary time and this moment:

$$\tau_{conv} = t_b - t. \tag{3.9}$$

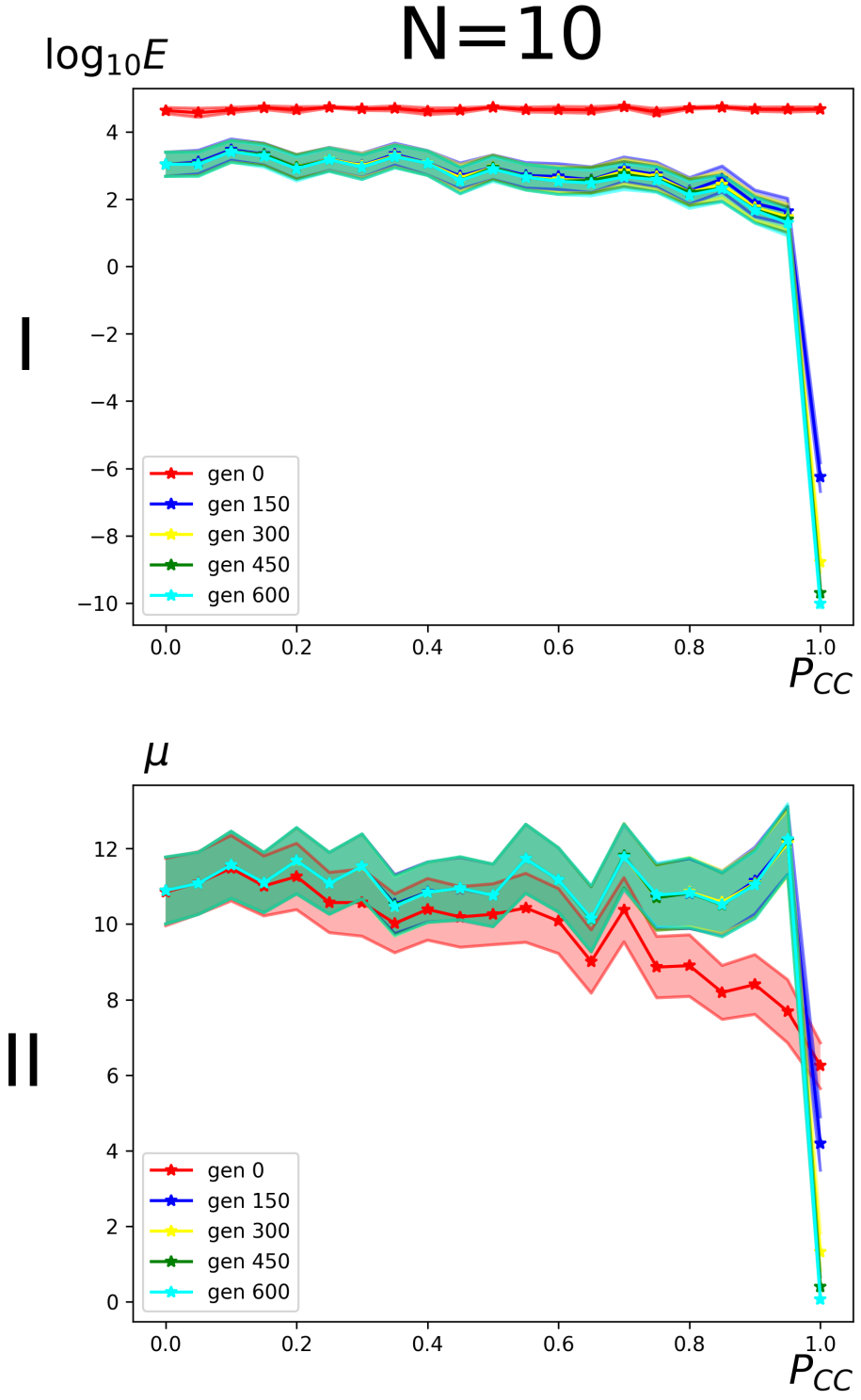
Note that if evolution has reached the baseline performance in the past, τ_{conv} is negative. For every generation of every run in the P_{mm} group we considered the position of the population on the μ-τ_{conv} plane. The density of the points is shown in Figure 3.5, right panel. It can be seen that convergence ($\tau_{conv} \leqslant 0$) in the majority of cases occurred after discovering the $J = I$ morphology, and in almost all cases after coming within a Hamming distance of 1 from it. It can also be seen that even after the $J = I$ morphology is found, it took more than 50 generations for some runs to converge (compare with the maximum convergence time of about 200 (data not shown)). These results are consistent with the hypothesis 1, but suggest a the following clarification of hypothesis 0: approaching a morphology that admits modular control is usually a necessary but not sufficient condition for rapid convergence of modularity-biased control evolution.


N=3
log10E
I
0
−2
−4
−6
−8
gen 0
gen 150
gen 300
gen 450
gen 600
0.0
0.2
0.4
0.6
0.8
1.0
PCC
μ
II
4
3
2
1
0
gen 0
gen 150
gen 300
gen 450
gen 600
0.0
0.2
0.4
0.6
0.8
1.0
PCC

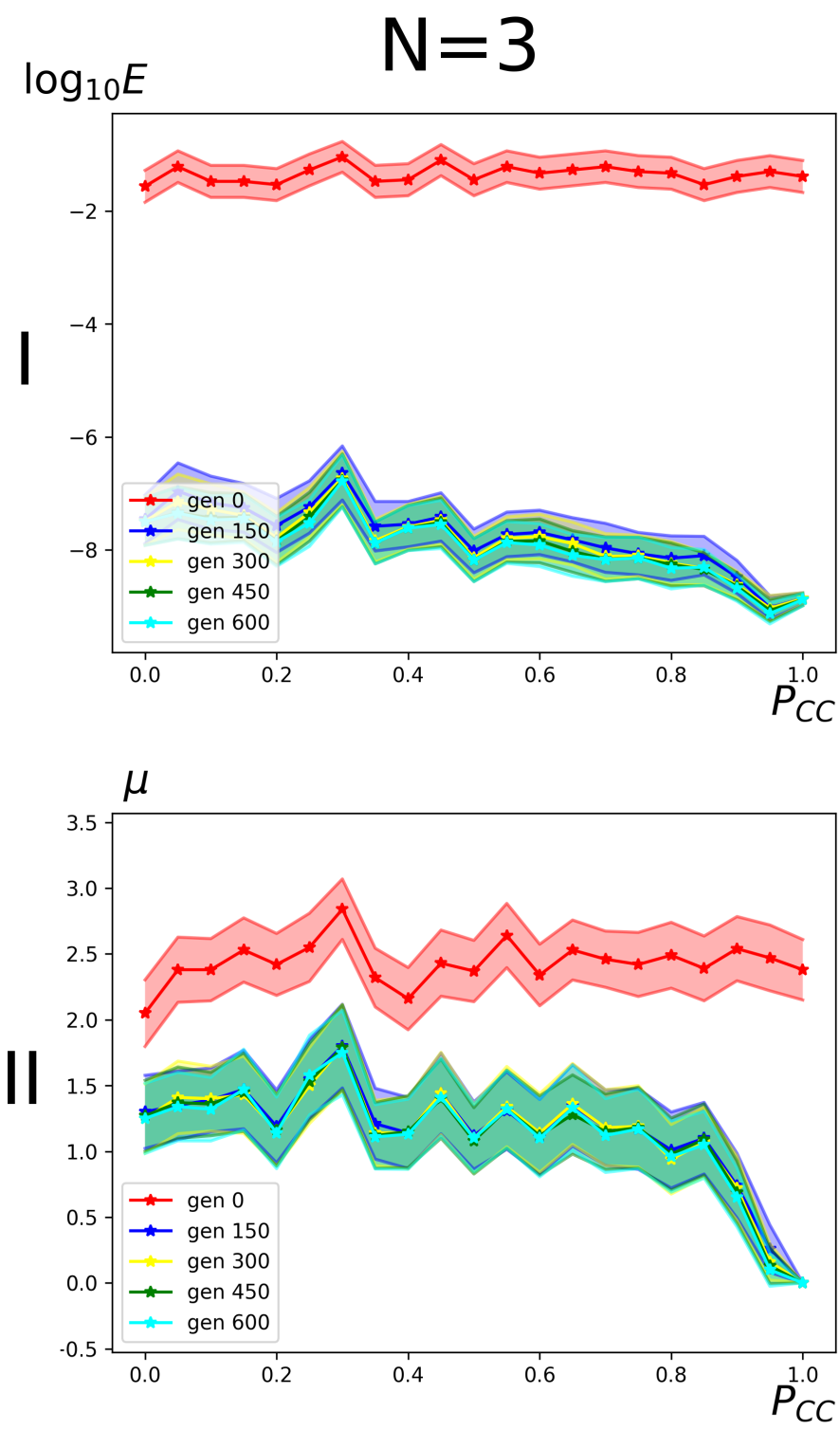
(A1)


N=5
$\log_{10}E$
I
0
−2
−4
−6
−8
−10
gen 0
gen 150
gen 300
gen 450
gen 600
0.0
0.2
0.4
0.6
0.8
1.0
P_{CC}
μ
II
6
5
4
3
2
1
0
gen 0
gen 150
gen 300
gen 450
gen 600
0.0
0.2
0.4
0.6
0.8
1.0
P_{CC}

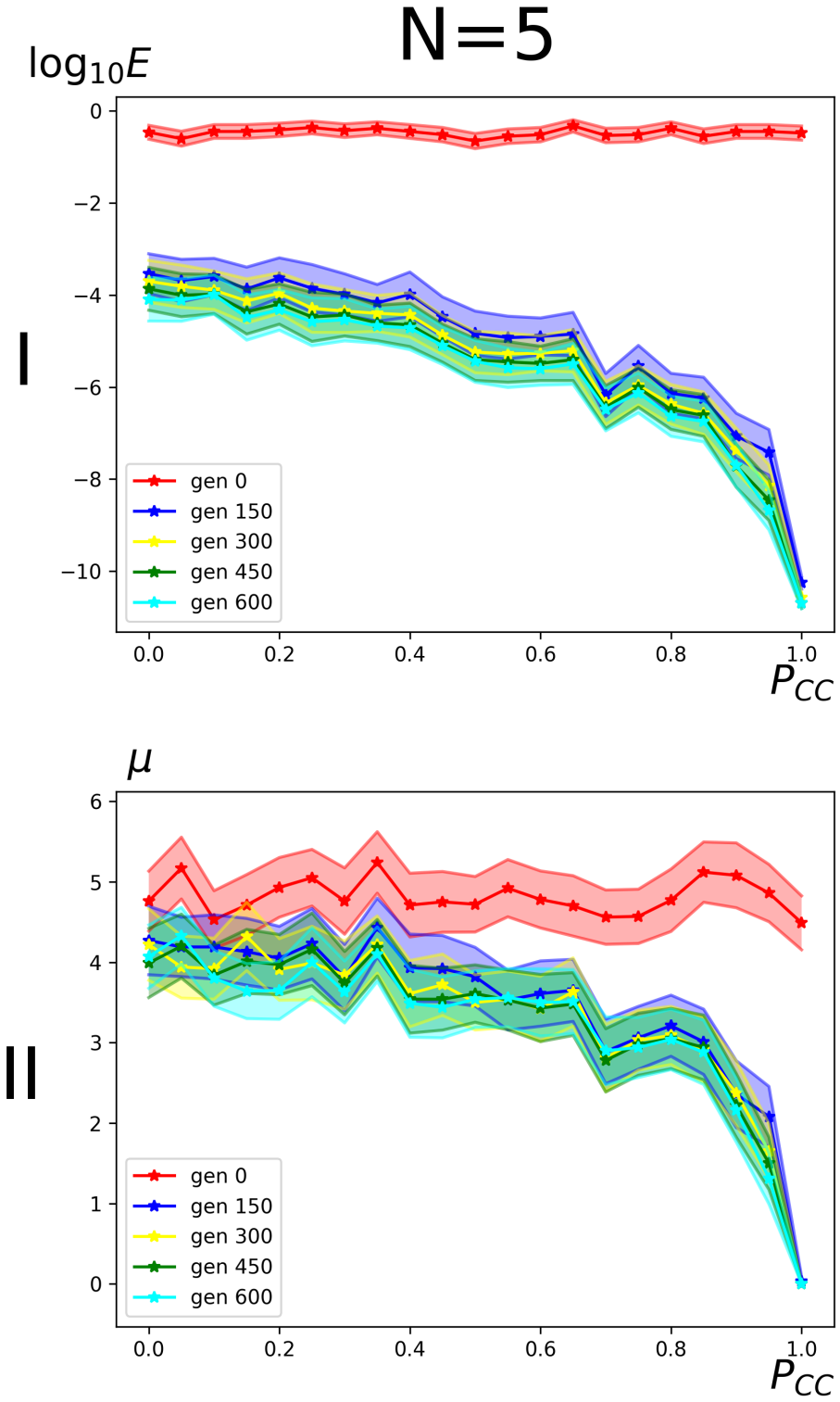
(A2)


N=10
log10E
I
4
2
0
−2
−4
−6
−8
−10
gen 0
gen 150
gen 300
gen 450
gen 600
0.0
0.2
0.4
0.6
0.8
1.0
PCC
μ
II
12
10
8
6
4
2
0
gen 0
gen 150
gen 300
gen 450
gen 600
0.0
0.2
0.4
0.6
0.8
1.0
PCC

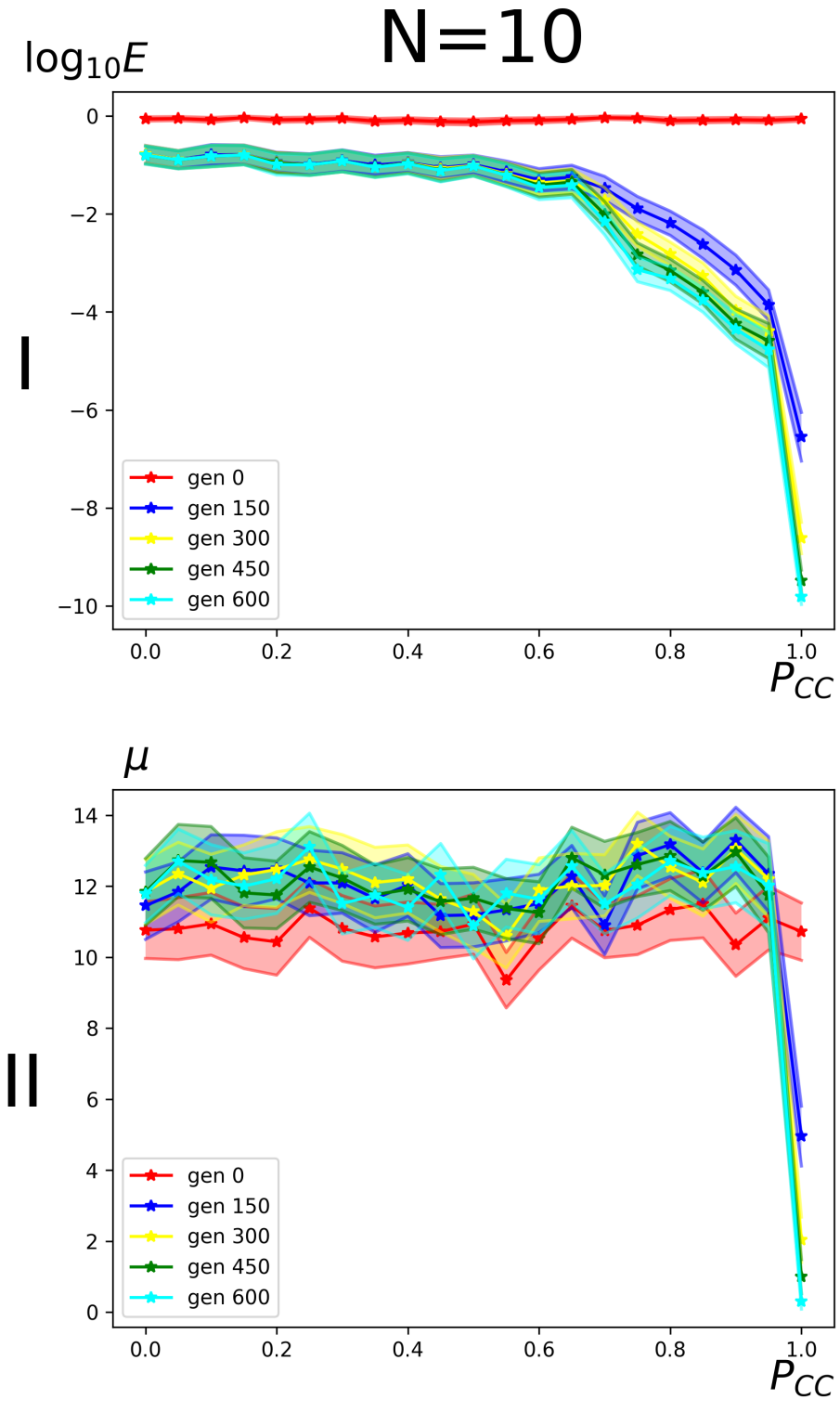
(A3)


N=3
$\log_{10}E$
I
−2
−4
−6
−8
gen 0
gen 150
gen 300
gen 450
gen 600
0.0
0.2
0.4
0.6
0.8
1.0
P_{CC}
μ
II
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
-0.5
gen 0
gen 150
gen 300
gen 450
gen 600
0.0
0.2
0.4
0.6
0.8
1.0
P_{CC}

(B1)


N=5
$\log_{10}E$
I
0
−2
−4
−6
−8
−10
gen 0
gen 150
gen 300
gen 450
gen 600
0.0
0.2
0.4
0.6
0.8
1.0
P_{CC}
μ
II
6
5
4
3
2
1
0
gen 0
gen 150
gen 300
gen 450
gen 600
0.0
0.2
0.4
0.6
0.8
1.0
P_{CC}

(B2)


N=10
$\log_{10}E$
I
0
−2
−4
−6
−8
−10
0.0
0.2
0.4
0.6
0.8
1.0
P_{CC}
gen 0
gen 150
gen 300
gen 450
gen 600
μ
II
14
12
10
8
6
4
2
0
0.0
0.2
0.4
0.6
0.8
1.0
P_{CC}
gen 0
gen 150
gen 300
gen 450
gen 600

(B3)

Figure 3.1: ***(A1-A3)*** *Parameters of populations as a function of probability of connection cost to be taken into account when comparing individuals, P_{CC}, with initial populations of dense networks. Columns (**pages in thesis version, A1-A3**) correspond to numbers of segments in Arrowbots N (see Table 3.1 for details about evolutionary algorithm's parameters). Top (I) row shows the average decimal logarithm of error $\log_{10} E$ and its 95% confidence interval computed using the data from 100 evolutionary runs; bottom (II) row shows the average minimal distance μ to the morphology $J = I$ across the stochastic Pareto front and its 95% confidence interval. Different lines represent values measured at different points of evolutionary time (see legend). It can be seen that the convergence to the morphology $J = I$ (corresponding to $\mu = 0$) coincides with the lowest observed errors and is reached reliably only for $P_{CC} = 1$.* ***(B1-B3)*** *Same, but with an initial population of sparse networks. This modification decreases the error more rapidly and approaches the $J = I$ morphology for all values of P_{CC}, not just $P_{CC} \approx 1$. However, the impact of initial population decreases as the task is scaled up.*

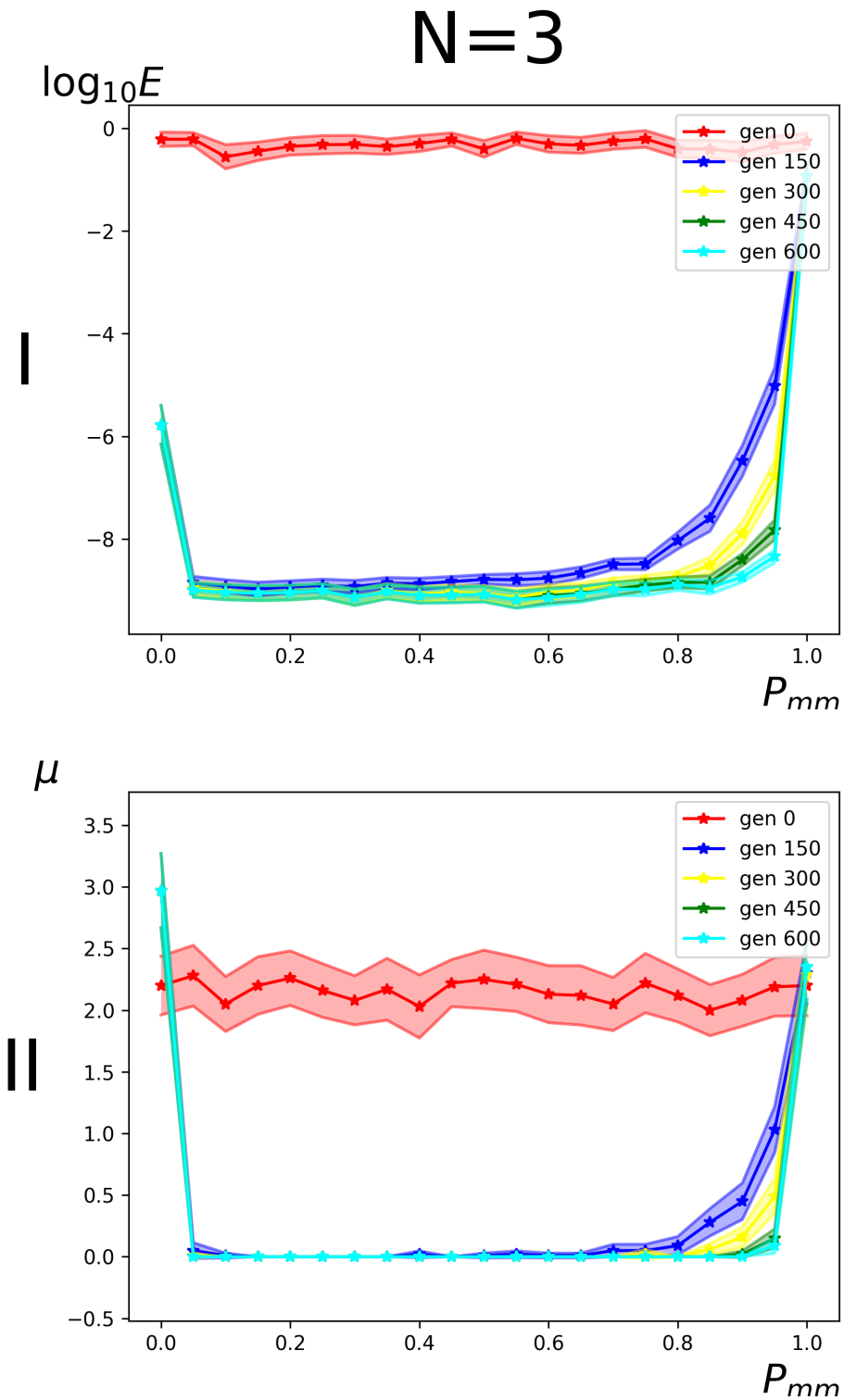
Next, we investigated whether the outcome of evolution depends on whether the connection cost objective is used. To do that, we investigated the dependence of the behavior of the system on the probability of connection cost to be taken into account when deciding on dominance, P_{CC} (Fig. 3.1A). Dependence was investigated for three values of robot size ($N = 3, 5, 10$) with the population size and maximum number of generations selected separately for each value in order to account for the disparity in task difficulty (Table 3.1). The populations were initialized with dense networks as described in Methods. For each value of N, we selected five equidistant temporal slices at which we measured two properties of the population – smallest achieved error E and the parameter μ. We varied P_{CC} across $\{0, 0.05, 0.1, ..., 1\}$ and observed that for all values of N there is a sharp decrease in both error and μ as P_{CC} approaches 1. This suggests that the presence of bias towards modularity is crucial for evolving the morphologies that admit modular control and for achieving rapid convergence of control evolution, consistent with hypotheses 4 and 5.

Additionally, we have found that the decrease in the error and μ becomes more pronounced as the task is scaled up. This suggests that for more complex tasks bias towards modularity is more crucial than for the simpler ones.

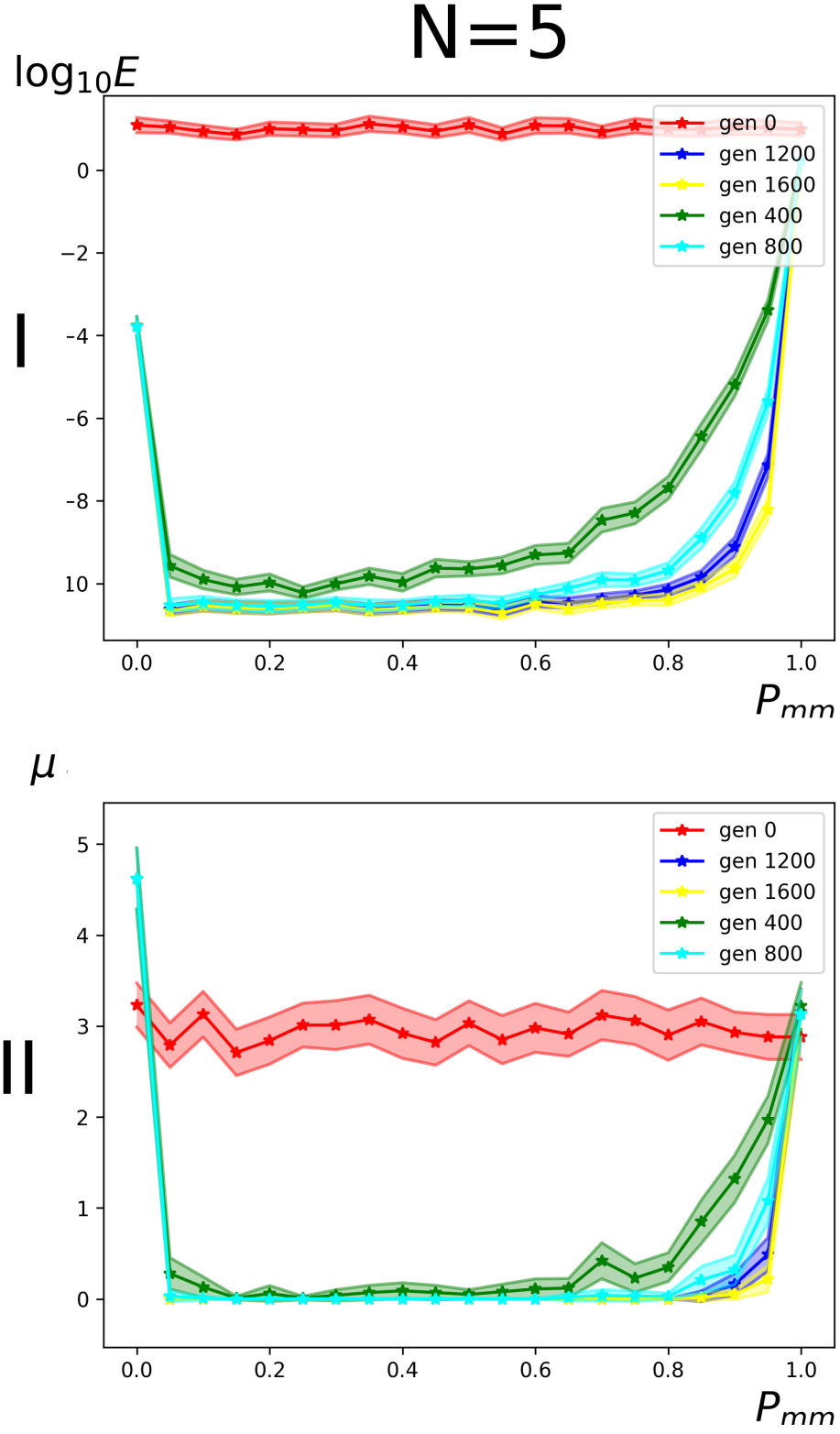
We also investigated whether the approach relies on the particular bi-objective performance and connection cost technique, or if it is agnostic with respect to the particular biasing technique. Previously we demonstrated that initializing the population with sparse networks can cause modularity to evolve even in the absence of the connection cost objective, albeit modularity tends to decrease at the later stages of evolution under such conditions [7] (*note - also provided as Section A.1*). This suggests that initializing the population with sparse networks can be an efficient way

to bias evolution towards modular networks, although the efficiency of the approach may decrease as the number of generations required to reach an optimal solution increases (e.g. for more complex tasks).

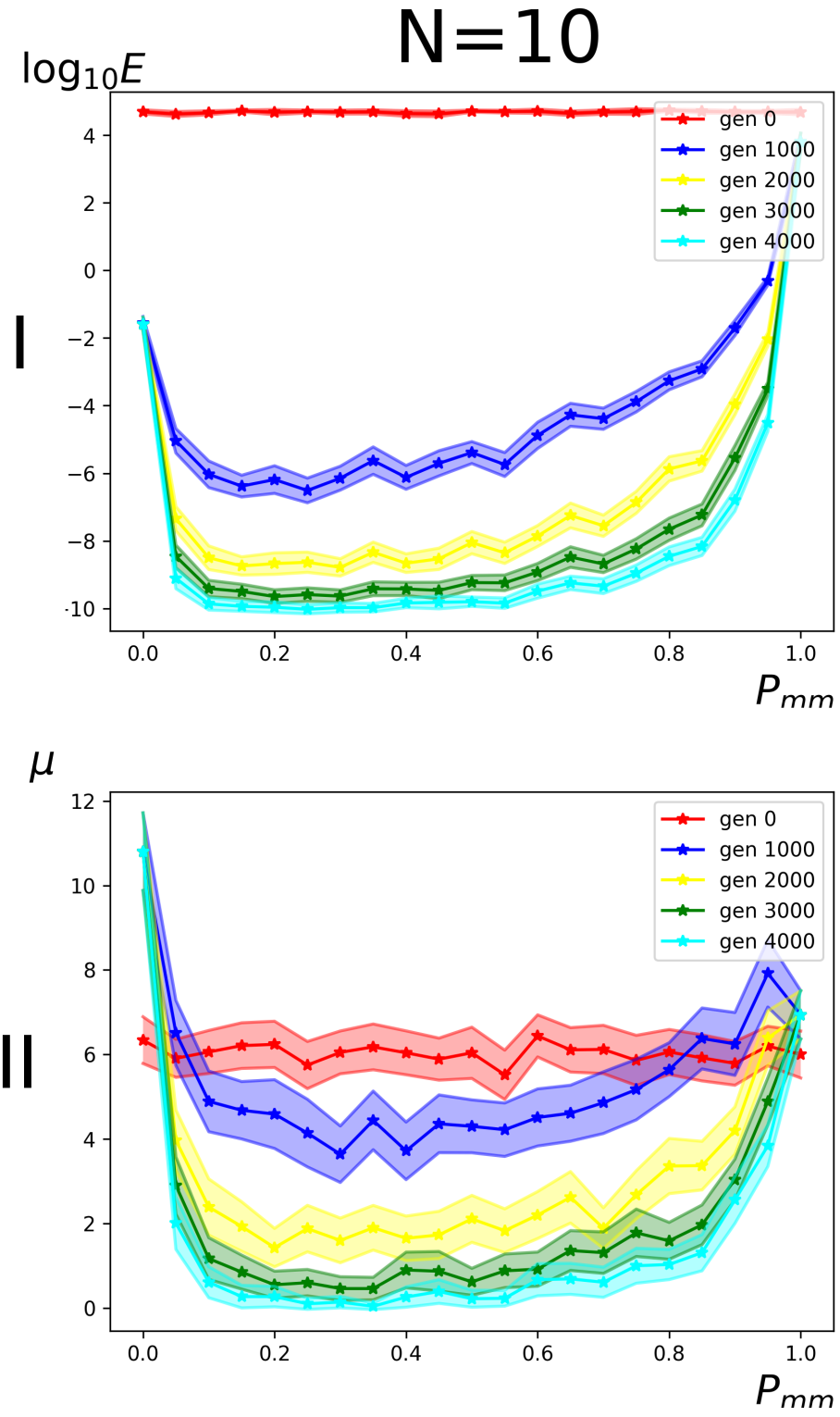
We repeated the experiments detailing the dependence of evolutionary dynamic on P_{CC} with initial populations of sparse networks and found that evolution exhibits smaller final errors and more rapid convergence for all values of P_{CC}, although the effect decreased as the task was scaled up (Fig. 3.1B). In particular, for $N = 3$ the final error for $P_{CC} = 0$ (connection cost not used) was comparable to the final error of the runs initialized with dense networks with $P_{CC} = 1$ (fully bi-objective approach). The runs with sparse populations also exhibited on average smaller minimal distance from the $J = I$ morphology to the Pareto front, μ. For greater values of N, the differences were qualitatively the same, but of magnitude that decreased with N. This is consistent with hypothesis 6.


N=3
$\log_{10}E$
I
gen 0
gen 150
gen 300
gen 450
gen 600
P_{mm}
μ
II
gen 0
gen 150
gen 300
gen 450
gen 600
P_{mm}

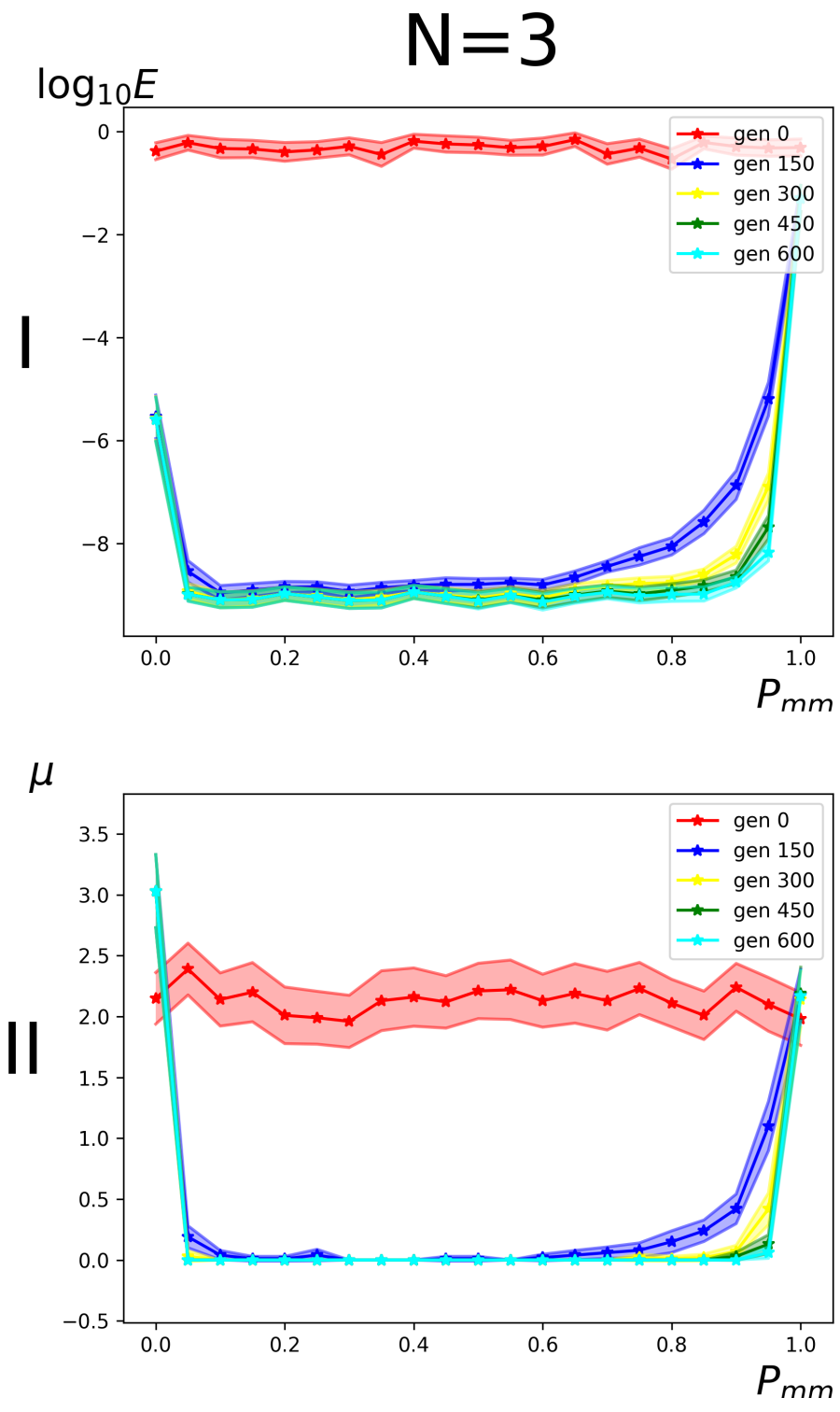
(A1)


N=5
log10E
I
gen 0
gen 1200
gen 1600
gen 400
gen 800
0
−2
-4
-6
-8
10
0.0
0.2
0.4
0.6
0.8
1.0
Pmm
μ
II
gen 0
gen 1200
gen 1600
gen 400
gen 800
5
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0
Pmm

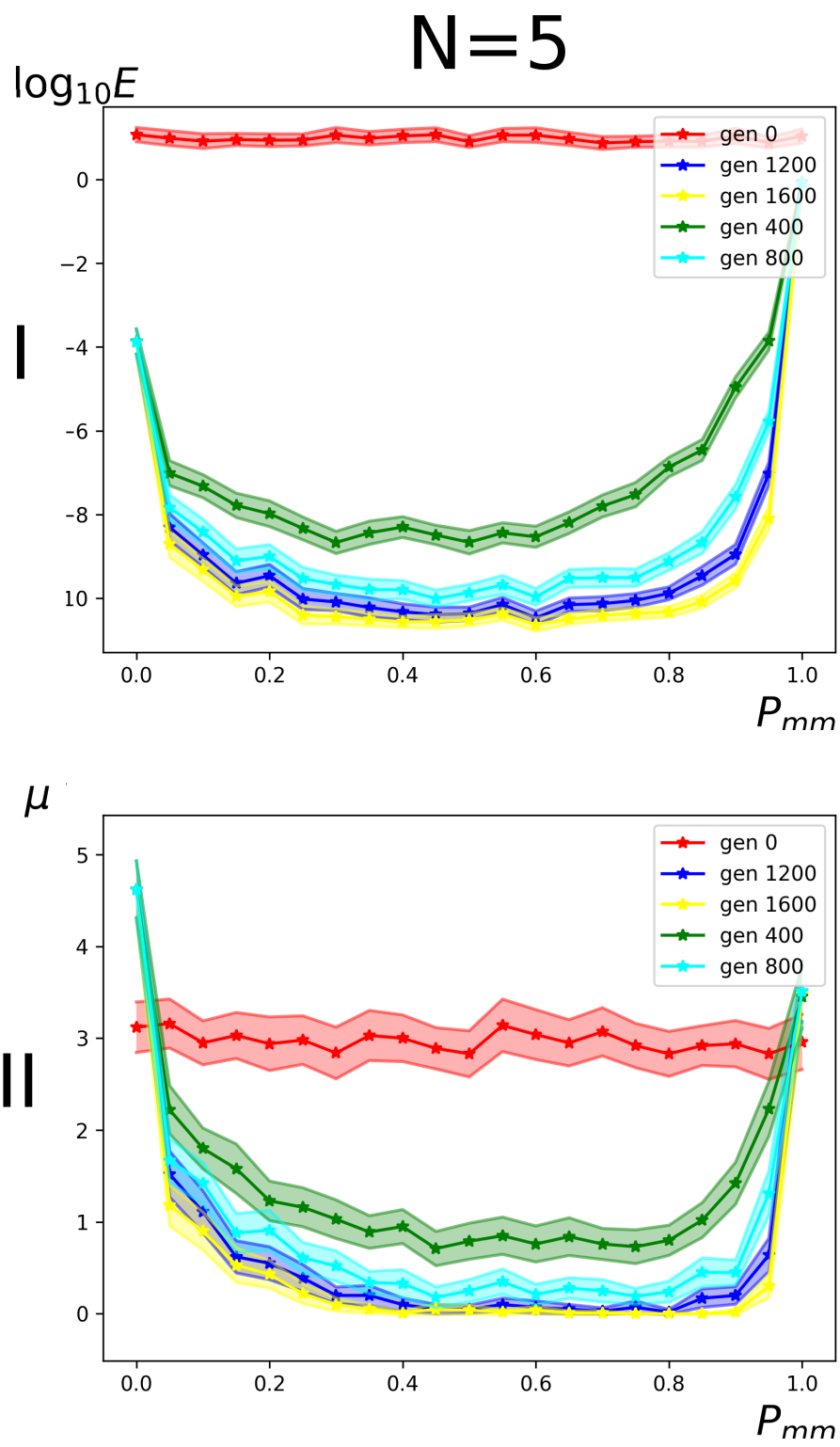
(A2)


N=10
log10E
I
gen 0
gen 1000
gen 2000
gen 3000
gen 4000
Pmm
μ
II
Pmm

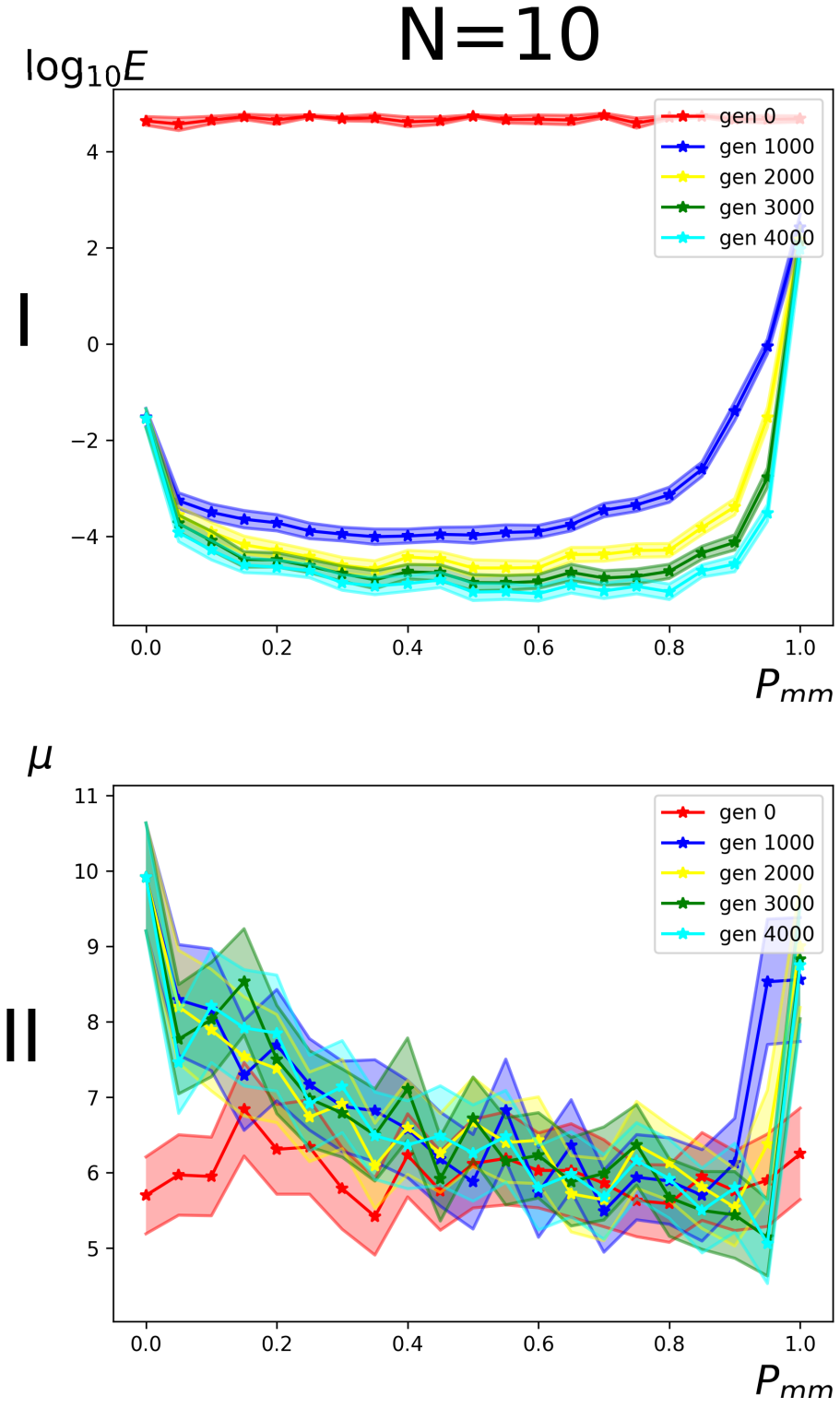
(A3)


N=3
$\log_{10}E$
I
gen 0
gen 150
gen 300
gen 450
gen 600
P_{mm}
μ
II
gen 0
gen 150
gen 300
gen 450
gen 600
P_{mm}

(B1)


N=5
log10E
I
gen 0
gen 1200
gen 1600
gen 400
gen 800
0
−2
-4
-6
-8
10
0.0
0.2
0.4
0.6
0.8
1.0
Pmm
μ
II
gen 0
gen 1200
gen 1600
gen 400
gen 800
5
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0
Pmm

(B2)


N=10
$\log_{10}E$
I
gen 0
gen 1000
gen 2000
gen 3000
gen 4000
4
2
0
−2
−4
0.0
0.2
0.4
0.6
0.8
1.0
P_{mm}
μ
II
11
10
9
8
7
6
5
0.0
0.2
0.4
0.6
0.8
1.0
P_{mm}

(B3)

Figure 3.-3: ***(A1-A3)*** *Parameters of populations as a function of probability of mutation to be morphological P_{mm}. Columns (each shown on a separate page in the dissertation version) correspond to numbers of segments in Arrowbots N (see Table 3.1 for details about evolutionary algorithm's parameters). Top (I) row shows the decimal logarithm of error $\log_{10} E$ and its 95% confidence interval computed using the data from 100 evolutionary runs; bottom (II) row shows the minimal distance μ to the morphology $J = I$ across the Pareto front and its 95% confidence interval. Different lines represent values measured at different points of evolutionary time (see legend). It can be seen that the convergence to $J = I$ is reached for a wide range of P_{mm} values; however, this process occurs most rapidly for lower values of P_{mm}, resulting in lower errors being achieved earlier on. The effect get more pronounced as the task is scaled up.* ***(B1-B3)*** *Same, but with the morphological mutation replaced by a random jump in the morphospace. It can be seen that convergence does occur even if the space of morphologies is searched randomly, although the performance of this approach suffers more as the task is scaled up, compared to the mutation operator that moves a sensor by one segment.*

Finally, we investigated the relationship between the probability of mutation to be morphological, P_{mm} and the evolutionary behavior of the system (Fig. 3.-3A). We found that convergence of the morphology to $J = I$ occurs for most values of P_{mm} within $(0, 1)$, except for values that are very close to 0 or 1. However, lower values of P_{mm} lead to a more rapid error reduction and make the discovery of $J = I$ occur more rapidly than for higher values of P_{mm}. This is consistent with hypothesis 7. The data we currently have does not allow to reliably measure the optimal value of P_{mm}, but the value appears to not depend strongly on the number of segments N.

We also checked how the adaptation rate depends on the morphological mutation operator. To that end, we repeated the P_{mm} investigation with the mutation operator replaced by a random jump in the morphological space (Fig. 3.-3B). It can be seen that the $J = I$ morphology is still found and that evolution still reaches the baseline level, although it takes longer, especially as the task is scaled up. This suggests that the complexity of the control optimization task is sufficient to justify random search of morphology, especially for lower values of N. As the task is scaled up, however, the design of the morphological mutation operator starts to play a more significant role. Another observation is that if the morphological mutation is a random jump, then the number of generations required to perform the search in the morphological space increases and so does the optimal value of the probability P_{mm}.

3.7 Discussion

We have shown that, at least for one task and family of robot morphologies, it is possible to discover, via evolutionary search, a morphology admitting highly

modularized and successful control by evolving the morphology alongside the control in a particular manner (hypothesis 2). We found that biasing evolution towards modular controllers is crucial for finding such a morphology (hypotheses 4 and 5), that the change in implementation of the bias does not make prevent the morphology from being found (hypothesis 6) and that under such bias and once such a morphology is found, rapid convergence of control towards a modular solution is likely to occur (hypotheses 1 and 3). Additionally, we have found that slowing the morphological change relative to change in control is beneficial for the occurrence of the phenomenon (hypothesis 7), unless the morphological mutation operator is very inefficient (random jump).

Those results are consistent with our previous findings that morphology can profoundly influence the difficulty of evolving modular control and describe conditions that make modular control more likely to evolve. By supporting these findings, our results also indirectly support the idea of morphological computation [93], as they represent a case of radical simplification of control due to the choice of morphology.

These results can be used in engineering, in particular, in evolutionary robotics, where they may allow the designers to place a greater number of heterogeneous design variables under the control of evolution, while avoiding, through modularity, the requirement for a combinatorially large number of evaluation environments [18] and the problem of catastrophic forgetting [34] and thus retaining the capacity of evolution to find good-enough solutions in reasonable time. For example, these results may be of interest to engineers and researchers who wish to evolve whole physical agents, complete with morphology, circuitry and actuator design.

The results also have implications for biology, as living systems satisfy the basic

requirements of our approach to the evolution of modular control: the bias towards modular nervous systems is present in living systems in form of metabolic connection cost [26] and morphology is evolved alongside the control. Based on that, we speculate that morphological evolution might be a factor in evolution of modularity of nervous systems.

To formulate our approach to evolving modular control, we formalized the reformulation approach to problem solving [25, 113] (see Section 3.4). Within the reformulation framework, morphology is one example of a more general entity which we term driving variables, defined as design parameters that influence the search difficulty of finding the best overall values of the remaining (non-driving) variables. Finding such variables requires domain knowledge, but once they are found, they can be optimized to reduce the search difficulty of optimizing the remaining variables and thus "reformulate" the corresponding optimization problem. We extend the reformulation approach with a qualitative rule of thumb that suggests that the driving variables can be found by examining the influence of the design variables on the behavior of optimization of non-driving variables under biases. We call the resulting approach guided reformulation. The method for evolving modular control then follows from the hypothesis that morphology influences the effectiveness of evolution if a bias towards modularity is present. We speculate that the guided reformulation approach can be applied to reformulating a wider range of optimization problems than the problem of controlling embodied agents, possibly using optimization biases other than the bias towards modularity.

We briefly review one mechanism of optimization difficulty reduction, the division approach, in Introduction. We link it with solution modularity and explain why mor-

phology can influence this mechanism, thus establishing morphology as a candidate for a vector driving variable. Our set of testable hypotheses about the properties of the resulting optimization approach is based on this analysis.

We see three challenges for applying the reformulation approach and guided reformulation to a wider range of tasks. First, a possibility of premature convergence arise if the method is used with a definition of optimization difficulty that can mistakenly assign low difficulty to driving variable values that prohibit convergence of optimization of non-driving variables to good-enough solutions. Our instantiation of reformulation based on morphology and modularity-biased control optimization uses such a definition, and we plan to correct that in our future work using methods similar to morphological protection [21]. Second, despite the guidance provided by the guided reformulation rule-of-thumb, the identification of driving variables currently must be performed in an *ad hoc* manner by a human designer. Systematizing and/or automating the identification of driving variables will be the focus of our future studies. Third, for more complex tasks, a good mutation operator for driving variables is required. At present, there is no systematic way to assess or design such operators.

See the following GitHub repository for all the materials and instructions required to reproduce our results: https://github.com/abernatskiy/reformulation2018

3.8 Appendix A. Baseline performance for an arbitrary morphology

Theorem: It is possible to construct a linear controller satisfying equation (3.4) that reduces the ideal error (equation (3.6)) to zero for every valid sensor attachment

matrix J.

Proof: Substituting equation (3.3) and equation (3.2) into equation (3.4), we get

$$\begin{aligned} \dot{r} &= W\mathbf{s} + Yr = WT - WJKr + Yr \\ &= WT + (Y - WJK)r. \end{aligned} \tag{3.10}$$

At the equilibrium $r = K^{-1}T$ and

$$\begin{aligned} &WT + (Y - WJK)K^{-1}T = 0, \\ &Y = W(J - I)K. \end{aligned} \tag{3.11}$$

Assuming this condition holds, equation (3.10) simplifies to

$$\begin{aligned} \dot{r} &= WT + (W(J - I)K - WJK)r \\ &= WT - WKr. \end{aligned} \tag{3.12}$$

Consider a controller with $W = I$: a matrix with no values outside of $\{-1, 0, 1\}$. The Jacobian of the system then becomes $-K$, making the system stable.

Next, we verify that for each such controller, a matrix Y exists such that all its elements are in $\{-1, 0, 1\}$. From equation (3.11),

$$Y = (J - I)K. \tag{3.13}$$

Each row of the matrix $J - I$ contains at most one entry equal to -1 and one equal to 1, with the rest of entries being zeros. Multiplying that by the ith column of K will produce a sum of all entries to the right of the ith field, including the ith field itself. There will be at most two nonzero components in this sum, one equal to -1 and one

equal to 1. Such a sum necessarily lies in $\{-1, 0, 1\}$. ■

The Jacobian for all controllers constructed in this way is the same, thus the temporal trajectories of the systems are the same. We verified that this is true by evaluating the controllers constructed as described above for 100000 randomly picked morphologies for the three values of $N = 3, 5, 10$. Within each group, all logarithms of the pointing error equation (3.7) coincided up to tenth decimal point: $\log_{10} E(N = 3) = -6.9408767850$, $\log_{10} E(N = 5) = -6.0563588899$ and $\log_{10} E(N = 10) = -5.3115293183$.

We will use these values to define a baseline for controller performance. Such definition has the following property: given N, we can guarantee that for any morphology there exists a controller that achieves the pointing error no greater than the baseline.

References for chapter 3

[1] W Ross Ashby. *Design for a Brain: The origin of adaptive behavior*. 2nd. London: Chapman & Hall Ltd, 1960.

[2] Robert Axelrod et al. "The evolution of strategies in the iterated prisoner's dilemma". In: *The dynamics of norms* (1987), pp. 1–16.

[3] Carliss Y Baldwin and Kim B Clark. "Modularity in the design of complex engineering systems". In: *Complex engineered systems*. Springer, 2006, pp. 175–205.

[5] Anton Bernatskiy and Josh Bongard. “Choice of robot morphology can prohibit modular control and disrupt evolution”. In: *Proceedings of the 14th European Conference on Artificial Life*. MIT. 2017, pp. 60–67.

[7] Anton Bernatskiy and Joshua C Bongard. “Exploiting the relationship between structural modularity and sparsity for faster network evolution”. In: *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM. 2015, pp. 1173–1176.

[9] Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[14] Josh C Bongard et al. “Evolving robot morphology facilitates the evolution of neural modularity and evolvability”. In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM. 2015, pp. 129–136.

[15] Josh Bongard, Victor Zykov, and Hod Lipson. “Resilient machines through continuous self-modeling”. In: *Science* 314.5802 (2006), pp. 1118–1121.

[18] Collin K Cappelle et al. “Morphological Modularity Can Enable the Evolution of Robot Behavior to Scale Linearly with the Number of Environmental Features”. In: *Frontiers in Robotics and AI* 3 (2016), p. 59.

[19] Sean B Carroll. “Chance and necessity: the evolution of morphological complexity and diversity”. In: *Nature* 409.6823 (2001), pp. 1102–1109.

[21] Nick Cheney et al. “Scalable Co-Optimization of Morphology and Control in Embodied Machines”. In: *arXiv preprint arXiv:1706.06133* (2017).

[25] Berthe Y Choueiry, Yumi Iwasaki, and Sheila McIlraith. "Towards a practical theory of reformulation for reasoning about physical systems". In: *Artificial Intelligence* 162.1-2 (2005), pp. 145–204.

[26] Jeff Clune, Jean-Baptiste Mouret, and Hod Lipson. "The evolutionary origins of modularity". In: *Proc. R. Soc. B.* Vol. 280. 1755. The Royal Society. 2013, p. 20122863.

[31] Jeremy Draghi and Günter P Wagner. "Evolution of evolvability in a developmental model". In: *Evolution* 62.2 (2008), pp. 301–315.

[32] Karl Duncker and Lynne S Lees. "On problem-solving." In: *Psychological monographs* 58.5 (1945), p. i.

[33] Peter Durr, Dario Floreano, and Claudio Mattiussi. "Genetic representation and evolvability of modular neural controllers". In: *IEEE Computational Intelligence Magazine* 5.3 (2010), pp. 10–19.

[34] Kai Olav Ellefsen, Jean-Baptiste Mouret, and Jeff Clune. "Neural modularity helps organisms evolve to learn new skills without forgetting old skills". In: *PLoS Comput Biol* 11.4 (2015), e1004128.

[36] Carlos Espinosa-Soto and Andreas Wagner. "Specialization can drive the evolution of modularity". In: *PLoS Comput Biol* 6.3 (2010), e1000719.

[38] Adrienne L Fairhall et al. "Multiple timescales of adaptation in a neural code". In: *Advances in neural information processing systems*. 2001, pp. 124–130.

[40] Dario Floreano and Francesco Mondada. "Automatic creation of an autonomous agent: Genetic evolution of a neural network driven robot". In: *Proceedings of*

the third international conference on Simulation of adaptive behavior: From Animals to Animats 3. LIS-CONF-1994-003. MIT Press. 1994, pp. 421–430.

[45] Michelle Girvan and Mark EJ Newman. "Community structure in social and biological networks". In: *Proceedings of the national academy of sciences* 99.12 (2002), pp. 7821–7826.

[49] Jeff Hawkins and Sandra Blakeslee. *On intelligence: How a new understanding of the brain will lead to the creation of truly intelligent machines*. Macmillan, 2007.

[51] Geoffrey E Hinton and Steven J Nowlan. "How learning can guide evolution". In: *Complex systems* 1.3 (1987), pp. 495–502.

[54] Julian Huxley. *Evolution the modern synthesis*. George Allen and Unwin, 1942.

[56] Nadav Kashtan and Uri Alon. "Spontaneous evolution of modularity and network motifs". In: *Proceedings of the National Academy of Sciences of the United States of America* 102.39 (2005), pp. 13773–13778.

[57] Stefan J Kiebel et al. "Recognizing sequences of sequences". In: *PLoS computational biology* 5.8 (2009), e1000464.

[58] Jon Kleinberg and Eva Tardos. *Algorithm design*. Pearson, 2005.

[59] Günther Knoblich, Stellan Ohlsson, and Gary E Raney. "An eye movement study of insight problem solving". In: *Memory & cognition* 29.7 (2001), pp. 1000–1009.

[60] Loizos Kounios et al. "Resolving the paradox of evolvability with learning theory: How evolution learns to improve evolvability on rugged fitness landscapes". In: *arXiv preprint arXiv:1612.05955* (2016).

[63] Sam Kriegman et al. “Evolving Spatially Aggregated Features from Satellite Imagery for Regional Modeling”. In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2016, pp. 707–716.

[76] Hod Lipson et al. “On the origin of modular variation”. In: *Evolution* 56.8 (2002), pp. 1549–1556.

[77] Paul Martin, Paul Patrick Gordon Bateson, and Patrick Bateson. *Measuring behaviour: an introductory guide*. Cambridge University Press, 1993.

[78] Maja J Matarić. “Behavior-based robotics as a tool for synthesis of artificial behavior and analysis of natural behavior”. In: *Trends in cognitive sciences* 2.3 (1998), pp. 82–86.

[80] Risto Miikkulainen et al. “Evolving Deep Neural Networks”. In: *arXiv preprint arXiv:1703.00548* (2017).

[85] Mark EJ Newman. “Modularity and community structure in networks”. In: *Proceedings of the national academy of sciences* 103.23 (2006), pp. 8577–8582.

[87] Stefano Nolfi and Dario Floreano. “Coevolving predator and prey robots: Do "arms races" arise in artificial evolution?” In: *Artificial life* 4.4 (1998), pp. 311–335.

[90] Steven Orzack, Steven Hecht Orzack, and Elliott Sober. *Adaptationism and optimality*. Cambridge University Press, 2001.

[92] Rolf Pfeifer and Josh Bongard. *How the body shapes the way we think: a new view of intelligence*. MIT press, 2006.

[93] Rolf Pfeifer, Fumiya Iida, and Gabriel Gómez. “Morphological computation for adaptive behavior and cognition”. In: *International Congress Series*. Vol. 1291. Elsevier. 2006, pp. 22–29.

[98] Christopher D Rosin and Richard K Belew. “New methods for competitive coevolution”. In: *Evolutionary computation* 5.1 (1997), pp. 1–29.

[107] Ricard V Solé and Pau Fernández. “Modularity" for free" in genome architecture?” In: *arXiv preprint q-bio/0312032* (2003).

[110] Kenneth O Stanley and Risto Miikkulainen. “A taxonomy for artificial embryogeny”. In: *Artificial Life* 9.2 (2003), pp. 93–130.

[113] Robert J Sternberg and Janet E Davidson. *The nature of insight*. The MIT Press, 1995.

[114] Strabo. *Geography*. 1903 translation by Hamilton, H.C. Esq. and Falconer, W. M.A., retrieved from the Perseus Digital Library 2017-10-30, reference Strab. 8.7.1. 23AD.

[115] Nam P Suh. *The principles of design*. 6. Oxford University Press on Demand, 1990.

[117] Niko Tinbergen. “On aims and methods of ethology”. In: *Ethology* 20.4 (1963), pp. 410–433.

[119] Nachum Ulanovsky et al. “Multiple time scales of adaptation in auditory cortex neurons”. In: *Journal of Neuroscience* 24.46 (2004), pp. 10440–10453.

[120] Conrad H Waddington. “Canalization of development and the inheritance of acquired characters”. In: *Nature* 150.3811 (1942), pp. 563–565.

[122] Günter P Wagner, Mihaela Pavlicev, and James M Cheverud. “The road to modularity”. In: *Nature Reviews Genetics* 8.12 (2007), pp. 921–931.

[124] Barry Wark, Adrienne Fairhall, and Fred Rieke. “Timescales of inference in visual adaptation”. In: *Neuron* 61.5 (2009), pp. 750–761.

[125] Richard A Watson and Eörs Szathmáry. “How can evolution learn?” In: *Trends in ecology & evolution* 31.2 (2016), pp. 147–157.

[126] John J Welch and David Waxman. “Modularity and the cost of complexity”. In: *Evolution* 57.8 (2003), pp. 1723–1734.

[127] Frank W Wicker et al. “Problem-reformulation training and visualization training with insight problems.” In: *Journal of Educational Psychology* 70.3 (1978), p. 372.

[129] Yuichi Yamashita and Jun Tani. “Emergence of functional hierarchy in a multiple timescale neural network model: a humanoid robot experiment”. In: *PLoS computational biology* 4.11 (2008), e1000220.

Chapter 4

Reformulating Scaffolding Fitness Functions To Address The Bootstrap Problem

This work is in preparation for publication in Genetic and Evolutionary Computation Conference 2019.

4.1 Abstract

Fitness, or utility functions are the primary way in which designer's intent is expressed in design-by-optimization techniques such as evolutionary robotics. For many tasks, "natural" fitness functions are constant within most of their domains, a property that severely impedes optimization. This is known as the bootstrap problem. One way to approach it is by scaffolding the optimization process, for example by adding auxiliary utility functions expressing potentially useful building blocks of the

solution. However, it is not trivial to combine the primary and auxiliary objectives, especially if some of them are in conflict. In this paper we study a simple metaoptimization algorithm that evolves scaffolding fitness functions, expressed as linear combinations of primary and auxiliary utility functions, in order to minimize the difficulty of the underlying evolutionary search. We apply the method to the problem of designing controllers for robots collectively assembling structures in zero gravity. We show that the approach with evolvable scaffolding outperforms the approaches with a fixed scaffolding fitness function. Additionally, we test a prediction made by the reformulation in optimization theory and show that a bias towards sparsity in the controller optimization negatively impacts the performance of the approach.

4.2 Intro

In design-by-optimization approaches such as evolutionary robotics and reinforcement learning, designer's intent is typically expressed as (or equivalently to) a fitness function on the set of possible designs. The function is then optimized using an appropriate method, such as a genetic algorithm. Performance of this approach depends heavily on the interplay between properties of the utility function and capabilities of the optimization method. For example, a simple hill climber is unlikely to find the global optimum of a deceptive non-convex fitness function. Fitness function design is one of the decisive factors in whether the design-by-optimization approach succeeds or fails.

For many task and environments, "natural", intuitive metrics of solution utility exist. For example, an easy way to quantify success of a mobile robot at locomotion

tasks is by measuring the distance traveled. For a wide range of tasks such "natural" utility functions are constant across most of their domains. Consider, for example, a mobile robot that is tasked with hauling an object from one point on the plane it inhabits to another. One natural utility function is the negative distance between the object and the target point at the end of the trial. If the robot is not close to the object at the beginning of the trial, most of its possible movements will never touch the object and the distance will stay exactly as it was at the beginning of the trial.

In such a fitness landscape any gradient-following algorithm with random initial conditions will, with high probability, pick up no fitness gradient. The most extreme case occurs when the fitness function is different from its base level only at the optima, the landscape shrewdly described in [51] as "needle in a haystack". For such landscapes, no structure of the function is exposed by any point measurement except for those that happen to be at the optima. It follows from No Free Lunch theorems [128] that no optimization algorithm can outperform random search under these conditions.

The situation does not become much better even if the fitness landscape is not as extreme and there is a small region of solution space with some gradient in it. If the representation of the solution has few dimensions, the search may happen to find this region by drifting or random restarts. However, assuming that the diameter of the region remains constant, the probability of that happening decreases exponentially with the number of dimensions in the solution representation. For complex tasks that require solution representations with many dimensions this type of landscape is as good as the needle in the haystack.

In evolutionary robotics this issue is known as **the bootstrap problem** [104].

It is extremely widespread in practical tasks and their "natural" fitness functions. A common feature of many such tasks is that they require a spatial or temporal coordination of multiple actions to make any progress. This includes many tasks in manipulation, manufacturing and some tasks in locomotion such as jumping. So-called multimodal tasks [47, 73, 53], in which the robot must exhibit different modes of behavior in different circumstances, are largely all in that category.

Bootstrap problem is similar to, but different from the more widely known problem of deception. In deceptive functions, the probability of a randomly selected solution candidate to land within an attractor of a "good-enough" optimum is much lower than the probability of it landing near a local optimum that is not "good-enough". In such a landscape, a simple greedy gradient follower is likely to get stuck at the local optimum and never progress from there. There are, however, methods for protecting the solutions that occupy different peaks than the champion (e.g. [102]), thus escaping the deception.

In contrast, for fitness functions that exhibit the bootstrap problem it is very difficult to discover *any* optimums, global or local.

Some functions can exhibit both problems. If most peaks of a function that exhibits the bootstrap problem are not "good-enough", any peak-seeking algorithm will be more likely to find one of these than one of the "good-enough" peaks. An example of such a function is a locomotion of a tetrapod robot with a directly encoded controller towards a goal in a deceptive labyrinth: any locomotion requires an unlikely coordination of actuator movement, and additionally there are many ways to locomote that bring the robot closer to the target without letting it reach the target.

The common feature of deception and the bootstrap problem is that the hypervol-

ume of attractors of "good-enough" solutions is small compared to the hypervolume of the less desirable subsets of the search space. This makes many methods designed for attacking one of these problems also applicable to the other and vice versa.

Due to the fundamental difficulty of the bootstrap problem all approaches that attack it necessarily must use some external knowledge about the structure of the task. The knowledge might come in form of a meaningful distance between behaviors (e.g. evolution of upright locomotion in [70]) or as an ability to decompose evaluation into multiple time scales, effectively increasing the number of evaluations by a large factor [51, 62], or in some other form.

The most straightforward approach is to utilize the available *a priori* knowledge about useful building blocks of the solution. This approach called **scaffolding** is arguably the most common way to attack the bootstrap problem. In the previous example with the robot tasked with hauling an object, scaffolding might involve adding an objective of minimizing the minimum distance between the robot and the object. In this case the robots that get closer than others to the object over their lifetime will get a reproductive advantage, until eventually a robot that touches the object will emerge, thus reaching the region of the solution space where the gradient of the fitness function is not zero.

Although scaffolding takes many forms, in this paper we will focus on scaffolding based on auxiliary objectives or fitness functions, similar to the example above. Auxiliary fitness functions express designer's domain knowledge of potentially useful building blocks of the solution. These functions are typically simpler to optimize, in a sense that they exhibit less bootstrap issues.

In this type of scaffolding, multiple auxiliary fitness functions are often useful.

Consider a variant of the object hauling task described above in which the robot is equipped with a manipulator, the object is a raw egg and the main (or *ultimate*) fitness function is now low if the egg breaks. Optimizing the auxiliary objective of closing in to the object will likely yield many solutions in which the robot approaches the egg too quickly, collides with it and breaks it; or solutions in which the egg survives the collision and rolls away. Solutions in which the robots closes in to the egg at a reasonable speed and grasps it will be extremely rare, again creating a "needle in a haystack" situation. To mend this, more auxiliary objectives corresponding to known useful building blocks are added. In this example, such objectives might include an auxiliary fitness function that penalizes high velocity approaches to objects and another one that rewards grasping.

Some objectives, such as the objective of approaching the egg and doing so at the slowest possible speed, might to some extent conflict with each other. As a result, combining such auxiliary objectives is a nontrivial task. A variety of methods have been proposed, including a simple linear combination [89, 53], gradual addition of subtasks into a single fitness objective [50, 11] and multiobjective optimization [53, 83]. However, all previous approaches fall into one of the two categories: they either adopt a diversity-based, Pareto multiobjective approach to exploring the search space, or they use a particular, human-designed way of combining the auxiliary objectives. In the former case the search becomes prone to the dimensionality curse; in the latter the way in which the objectives are combined introduces a bias that can be detrimental if the auxiliary objectives conflict.

In this paper we study a simple metaoptimization approach that evolves scaffolding fitness functions, represented as linear combinations of primary and auxiliary

fitness functions with evolvable coefficients. By letting the coefficients vary, the approach avoids the bias introduced by setting them to a constant value.

We apply the approach to a practically important task of assembling structures in zero gravity with a fleet of autonomous, tethered robots. We show that for this task it outperforms the search based on a fitness functions in which the coefficients are fixed (both by human and by random selection).

Method searches for a scaffolding fitness function that makes optimizing the primary fitness function easier; in this way it is an instance of reformulation in optimization [6] (*note - also provided as Chapter 3*). In this case, the coefficients of the linear combination are the variables influencing the difficulty of the underlying optimization process. Following [6] (*note - also provided as Chapter 3*) we refer to such variables as *driving variables*. One prediction of the reformulation framework is the heuristics of biased reformulation: if a bias is applied to the underlying optimization process and there is domain knowledge that suggests that the combination of this bias and some value of the driving variables drastically reduces the difficulty of the underlying optimization process, then adding such a bias is likely to increase the convergence rate of the metaoptimization. Conversely, if domain knowledge suggests that no value of driving variables will make optimization substantially easier under this bias, addition of the bias is likely to decrease the convergence rate.

It is well known [76, 26, 7] (*note - [7] is also provided as Section A.1*) that bias towards sparsity can substantially reduce the difficulty of certain optimization problems. A lot of open questions remain regarding the properties that distinguish the problems that are simplified by such a bias from those that are not. However, existing domain knowledge suggests that those properties have to do with the internal

structure of the task [26], in particular admissibility of modular solution [5] (*note - also provided as Chapter 2*).

Changing a scaffolding fitness functions does not change the task, but rather it changes the way the task is approached by the optimization algorithm. It can be inferred that there is no scaffolding fitness function that makes the search for a solution substantially easier with the bias towards sparsity than it was without. Based on that we predict that adding such a bias would hurt the convergence rate of the scaffolding fitness function evolution and confirm this prediction.

4.3 Methods

All materials required to reproduce this work can be found at https://github.com/abernatskiy/walter.

4.3.1 Task and robot

We consider a task of assembling a structure consisting of heavy parts that float in a zero gravity, frictionless environment. The assembly is performed by a fleet of lightweight robots equipped with thrusters, reaction control wheels and adhesive surfaces that can attach to the parts.

From a practical standpoint we're interested in evolving control strategies for lightweight robots. This approach is attractive because it reduces the mass that need to be launched, but large ratios of masses of parts to masses of robots makes it difficult to change both linear and, to a smaller extent, rotational velocities of parts using thrusters. We intentionally make this limitation more severe by modeling finite

fuel and saturatable reaction control wheels: thrusters can only deliver a finite amount of linear momentum over the lifetime of the robot and the rotational momentum that the reaction control wheels can deliver is similarly limited.

To make the robots capable of overcoming this limitation, they are also equipped with tethers and shock absorbers that enable them to pull the parts together and reduce the relative velocity of parts when they are in close proximity. The intention is to approximate the performance of human astronaut during a EVA: versatile limbs capable of pushing and pulling on objects enable the astronaut to change their relative position and orientation without spending any fuel. Tethers provide means to pull on objects separated by significant distances; shock absorbers provide pushing.

In present work we use a simulation based on Pyrosim* to study a variant of such a system that uses six spherical robots that form three pairs. Each robot has sensors of four kinds: proximity, light, proprioception of adhesive actuator and proprioception of tether. Robots within each pair are connected with an actuated tether.

Proximity sensors provide information about the position of the closest point of any physical object other than the body of the robot to the robot's center. Each sensor is sensitive only to objects within a spherical volume of radius four times that of the robot, centered around the same point. The sensor outputs three channels: (1) maximum penetration depth of any object into the spherical volume, divided by the diameter of the volume and (2,3) spherical angles θ and ϕ of closest point of any object to the center of the spherical volume. In practice this sensor can be implemented using several LiDARs.

Light sensors provide information about the proximity of remote light sources.

*https://ccappelle.github.com/pyrosim

Each light sensor and light source is associated with a "color": a feature of the radiation signal that allows to distinguish different kinds of it, such as wavelength or temporal modulation pattern. Output of each light sensor is given by

$$L_i^{(c)}(x, y, z) = \ln l^{(c)}(x, y, z)/l_0^{(c)}, \tag{4.1}$$

where $l^{(c)}$ is the total luminosity created by all light sources at (x, y, z):

$$l^{(c)} = \sum_j \left[\frac{1}{r_j^2} \text{ if } r_j > R \text{ else} \frac{1}{R^2} \right] \tag{4.2}$$

and $l_0^{(c)}$ is such luminosity at the sensor's position at the first time step of the simulation. In this equation is the distance from point (x, y, z) to the position of jth light source; R is the radius of the light source and summation is over all light sources of color c. No occlusion is modeled in present work. In practice such a sensing system may be implemented using temporally modulated gamma- or X-rays.

Within each pair, robots perceive the same "color" of light (see below); each pair of robots is associated with its own color.

Additionally, each robot is equipped with six proprioceptive sensors supplying the data about the state of the actuators:

- The number of objects currently adhesed to the adhesive surfaces.
- The tension of the tether.
- Reaction control wheels saturation data, one number for each axis.
- Thruster fuel gauge.

Actuators consist of a thruster fixed to the hull of the robot, three reaction control wheels, adhesive surface of the robot that can be turned on or off and tether that can be tensioned. Shock absorbers are currently implemented through adhesion: the robot's body inelastically absorbs kinetic energy if it is hit by the two parts from different sides while the adhesive sensor is active.

In total, each robot provides ten sensory channels and six actuator channels.

The control is by a standard continuous time-recurrent neural networks (CTRNNs) (e.g. [106]) with 10 sensor, 6 hidden and 6 motor neurons each. The output of each sensor neuron is the hyperbolic tangent of the output of the sensor it is attached to; outputs of all other neurons are given by

$$s_i^t = \tanh(\alpha_i s_i^{t-1} + \tau_i \sum_j w_{ji} s_j^{t-1}), \tag{4.3}$$

where s_i^t is the output of ith neuron at time step t, α_i and τ_i are the parameters of the neuron, w_{ji} is the connection weight between ith and jth neuron and the summation is over all connections terminating at the ith neuron (the neuron emitting each such connection is denoted as j).

Within the hidden layer, a connection can be made from any to any neuron, including self-recurrent connections. A bias neuron with a constant value of 1 is provided. The controllers for all robots in the fleet are identical.

Parts are modeled as two large cylinders (see table 4.1). On one of flat faces of each part we place three light sources of three different colors, corresponding to the colors of the robot pairs. Positioning error E_p is defined as the sum of distances between the light sources of the same color placed on different parts.

A screenshot of Pyrosim showing the fleet of the robots and its environment is

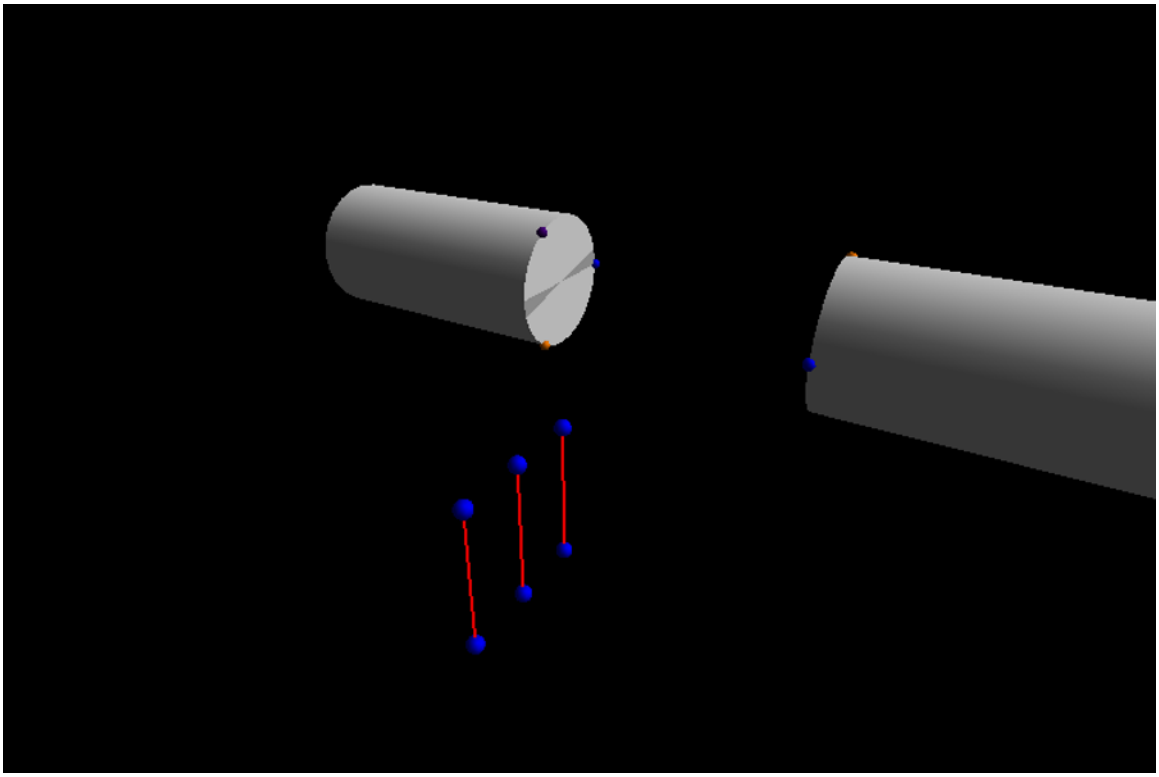


Figure 4.1: Screenshot of the simulator showing the fleet of robots and its environment.

Parameter	Value
Robot radius	0.25
Robot mass	1.0
Proximity sensor range	1.0
Light source radius	0.2
Part mass	1000.0
Part dimensions	10.0d2.5
Max evaluation time	52
Max RCW torque on each axis	0.1
RCW momentum budget	±0.05
Max thrust	1.0
Thruster momentum budget	10.0

Table 4.1: Physical and geometric parameters of the simulated system. All values are given in internal simulator units of their corresponding physical quantities.

shown at Figure 4.1.

4.3.2 Fitness

Primary fitness function expressed the following designer intent: "bring the parts together and align them without breaking anything". It is based on the positioning error, with a heavy penalty for high speed collisions:

$$F_P \equiv E_p^0 - \min_t(E_p^t) \,\text{if}\, v_{coll}^{max} < v_{coll}^{lim} \,\text{else}\, 0 \tag{4.4}$$

where E_p is the positioning error described in section Task, $E_p^0 = 375$ is its value at the first time step of the simulation, v_{coll}^{max} is the maximum relative velocity of the centers of mass of the colliding bodies in any collisions that occur over the simulation time and $v_{coll}^{lim} = 10$ simulation velocity units is its maximum allowed value.

In addition, we provide five auxiliary fitness functions, each of which expresses a useful (hopefully) solution building block:

Light following :

$$F_L \equiv \sum_{k=1}^{6} \frac{1}{T} \sum_t (L_i^{(c)})_k, \tag{4.5}$$

where $(L_i)_k$ is the output of light sensor of kth robot (characterized by id i and color c).

Obstacle avoidance :

$$F_X \equiv \sum_{k=1}^{6} \frac{1}{T} \sum_t p_0^k, \tag{4.6}$$

where p_0^k is the reading of the first channel (channel 0) of the proximity sensor of robot k.

Attachment :

$$F_A \equiv \sum_{k=1}^{6} \left[1 \text{ if } k\text{th robot has attached to any surface else } 0\right], \tag{4.7}$$

Fuel conservation :

$$F_F \equiv \sum_{k=1}^{6} \frac{1}{p_{lim}} \left[p_{lim} - \int_0^T f_{thruster}^k(t) dt \right], \tag{4.8}$$

where $f_{thruster}^k(t)$ is the force exerted on the body of kth robot by its thruster at

time t, T is the total time of the simulation and p_{lim} is the thruster's momentum budget.

Collision speed minimization :

$$F_C \equiv \frac{v_{coll}^{lim} - v_{coll}^{max}}{v_{coll}^{lim}} \text{ if } v_{coll}^{max} < v_{coll}^{lim} \text{ else } 0 \tag{4.9}$$

Total scaffolding fitness is a linear combination of all components:

$$F_S = F_P + \alpha_0 F_L + \alpha_1 F_X + \alpha_2 F_A + \alpha_3 F_F + \alpha_4 F_C, \tag{4.10}$$

where the coefficients $[\alpha_0 ... \alpha_4]$ were evolvable.

4.3.3 Genetic encoding and operators

Genome of a robotic fleet consists of two parts: the parameters of the scaffolding fitness and the parameters of the controller.

Parameters of the scaffolding fitness (see eq. 4.10) are encoded directly as a vector of five real numbers. Unless explicitly mentioned, their values are initialized as 2^i, where i is a random integer between -4 and 4, inclusively. Mutation of this part of the genome divided or multiplied a randomly selected value by 2, with a retry if the resulting value was outside of $[2^{-4}, 2^4]$. This enabled the evolution to increase or decrease the relative magnitude of different scaffolding fitness components with ease.

The second part of the genome encoded a CTRNN-based controller, common for all robots in the fleet. The first part of the genome encodes the parameters of each neuron except sensor neurons: τ_i, α_i and the initial state s_i^0, a total of $3 * (6 + 6) = 36$

values. The second part of the genome describes connections and weights within the three layers of the CTRNN: from sensor to hidden neurons, from hidden to hidden neurons and from hidden to motor neurons.

Mutation operator can perform one of the following actions:

- change a uniformly selected neuron parameter (with probability $p = 0.3$),
- change the weight of a uniformly selected connection ($p = 0.4$),
- add a new connection ($p = 0.15$),
- remove an existing connection ($p = 0.15$).

Whenever the mutation operator attempts to remove a connection from a network in which there are none or add a connection to a fully connected network, it is restarted.

A change in all continuous-valued parameters (neuron parameters and connection weights) is implemented with a simple Gaussian jump:

$$v \leftarrow v + \mathcal{N}(v, |v|). \tag{4.11}$$

Crossover is not used in this work.

When generating initial populations, the algorithm initializes the initial state of all neurons to -1, α's and τ's to 1. For connections, two types of initialization are explored:

Random , in which a connection is added between any pair of nodes in any two adjacent layers with a probability based of the average in-degree of 2.0 of the resulting network. Recurrent and non-recurrent connections are considered sep-

arately, leading to an average in-degree of 4.0 for hidden nodes in terms of combined connections.

Sparse , in which initially a single connection is added between a randomly selected pair of nodes [7] (*note - also provided as Section A.1*).

Self-loops were allowed. The initial value of the connection weight was chosen uniformly from $[-1, 1]$.

4.3.4 Evolutionary Algorithm

Our meta-evolutionary algorithm optimizes scaffolding fitness functions 4.10 in order to maximize the primary fitness 4.4. The algorithm is inspired by Age-Fitness Pareto Optimization technique for protecting innovation [102].

The population is divided into lineages. Each lineage is associated with a particular scaffolding fitness function (i.e., a vector of coefficients for 4.10). Additionally, age of each lineage is tracked.

At every generation, new population is formed as follows:

1. A set of nondominated genomes (i.e., genomes that provide optimal tradeoffs, see [102]) is found within the population, with respect to two objectives: maximization of the primary fitness 4.4 and minimization of age. This set, to which we will refer to as age-primary fitness Pareto front, forms the first part of the elite that is copied to the next generation without modifications.

2. Lineages that contain no individuals on the age-primary fitness Pareto front are discarded.

3. For each remaining lineage, one individual with a maximum scaffolded fitness function of the lineage is selected and copied to the new population without modifications, provided it is not already contained in there.

4. A lineage is selected at random among the remaining ones, and within the lineage a genome is selected for reproduction. Probability of reproduction is assigned to each genome of the lineage in a manner similar to fitness proportionate selection:

$$P^i = \frac{\epsilon + F_S^i}{\sum_i (\epsilon + F_S^i)}. \tag{4.12}$$

   Here, F_S^i is the scaffolding fitness of the genome 4.10 and $\epsilon = 1.0$ is a constant added to avoid divisions by zero and situations when a small improvement of fitness of a single individual in the lineage gives it a decisive reproductive advantage.

5. The selected genome is copied. Mutation operator (see section 4.3.3) is applied **to the controller part of the copy** and the result is added to the new population.

6. Steps 4 and 5 are repeated until the size of the new population reaches $N - 1$, where N is the fixed size of the population.

7. Once every m generations a new lineage is added into the population. All the lineages except the first initially contain a single genome. We experiment with three strategies of adding such a genome:

   (a) Adding a genome in which both the scaffolding fitness function and the controller are generated randomly. In this case each lineage uses the same

randomly generated scaffolding fitness function over its entire lifetime; we will refer to this strategy as *no turning*.

(b) Adding a genome every odd step is as in first strategy. At every even step a genome is selected as described in step 4. Its scaffolding fitness function is mutated as described in section 4.3.3 and it is added to the new population. This strategy enables the evolution to make slight adjustments to the scaffolding fitness function over the evolutionary history; we will refer to it as *smooth turning*.

(c) Adding a genome every odd step is as in first strategy. At every even step a genome is selected as described in step 4, but instead of mutating its fitness function it is discarded, a new one is generated randomly and assigned to the copy of the selected individual, which is then added to the new population. This strategy enables the evolution to make abrupt changes in the scaffolding fitness function; we will refer to it as *abrupt turning*.

8. If no new lineage was added, one more new individual is generated as described in steps 4 and 5 and added to the new population.

9. Age of all lineages is increased by 1. If a new lineage has been added, its age is reset to zero.

We also study a **sparsity biased** version of this algorithm. Similarly to [26], it uses multiobjective Pareto optimization of performance and connection cost to protect solution candidates which are sparser. This version of the algorithm differs from the one described above in two ways:

- At step 3, a set of nondominated individuals with respect to maximization of the scaffolding fitness function and minimization of the number of connections (scaffolding fitness - connection cost Pareto front) is found within the lineage. The set is copied to the new population without modifications.

- At step 4, a lineage is selected at random as before, but the genome is selected for reproduction among the genomes from the scaffolding fitness - connection cost Pareto front, with uniform probability.

In the preliminary runs we experimented with three values of the period of new lineage addition m: 25, 50 and 100. We found that the behavior of the algorithm is qualitatively the same within this set, so a value of $m = 50$ was chosen for subsequent use in all runs where the scaffolding fitness function was evolved. In all experiments described in this paper evolution runs for 6000 generations and the population size N is 100.

4.4 Results

In the first experiment we compare the performance of the algorithm to two controls in which evolution of scaffolding fitness functions is disabled (figure 4.2). Both controls are implemented by setting the period of lineage update to a value greater than the length of the run. With this setting, the algorithm turns into a simple GA with an elite of size 1 or 2[†] and fitness proportionate selection of parents for the rest of the population. In the first control group, the scaffolding fitness function was designed manually by setting all coefficients α_i to 1; in the second group, the

[†]1 if the champions of the ultimate fitness and the scaffolding fitness coincide and 2 otherwise.

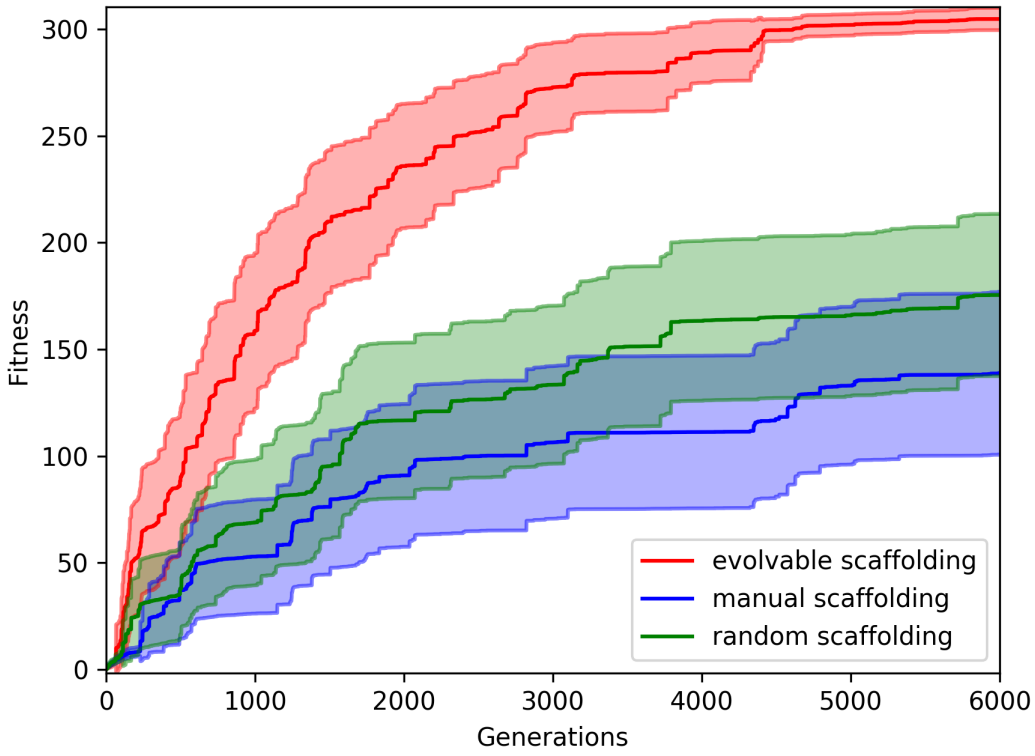


Figure 4.2: Performance of the evolutionary algorithm with evolution of scaffolding fitness function enabled (red line); disabled, with the coefficients alpha$_i$ selected randomly at the beginning of each run (green line); disabled, with the coefficients set manually to 1 (blue line). Error strips show 95% confidence interval of Student's t-distribution based on a sample of 50 runs.

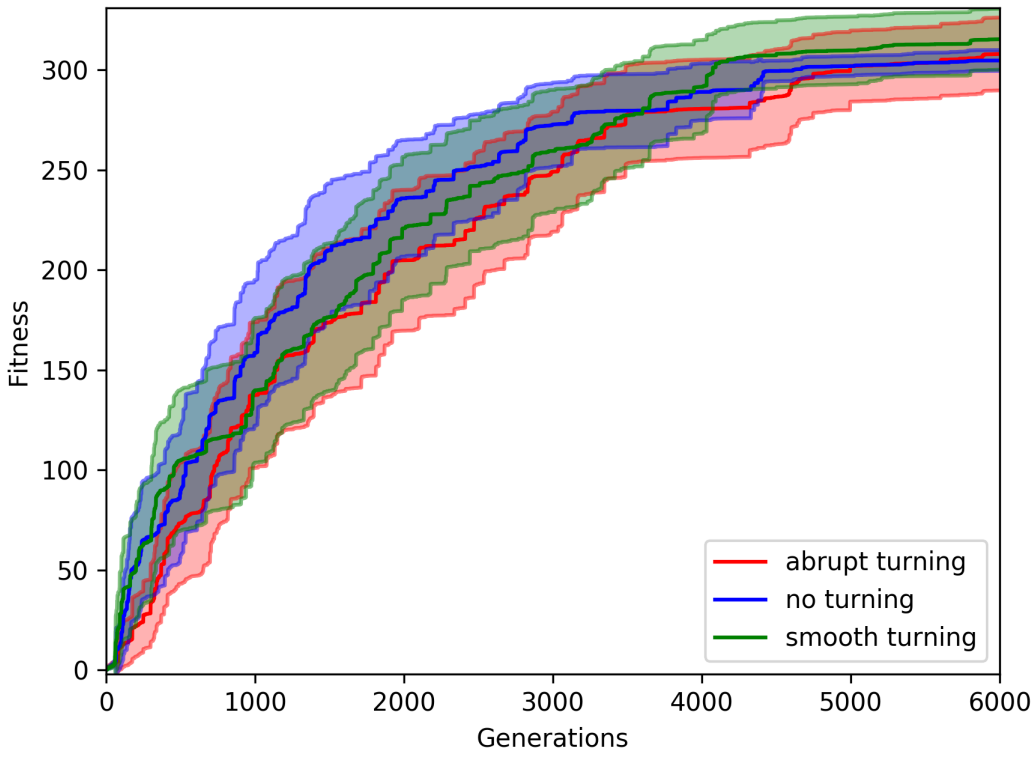


Figure 4.3: Performance of the scaffolding fitness function evolution for different strategies of adding new lineages (see section 4.3.4). Blue line corresponds to the no turning strategy (7(a) in text); green line corresponds to smooth turning (7(b) in text); red line corresponds to abrupt turning (7(c) in text). Error strips show 95% confidence interval of Student's t-distribution based on a sample of 50 runs.

coefficients were chosen randomly (as described in section 4.3.3) at the beginning of the run and never changed over its course. The approach with the evolvable fitness function outperformed both of these controls.

In the next experiment we investigated whether the performance of the algorithm depends on which strategy was used when adding new lineages (figure **??**). Interestingly, we did not find any substantial differences between the behavior of the system for the three strategies.

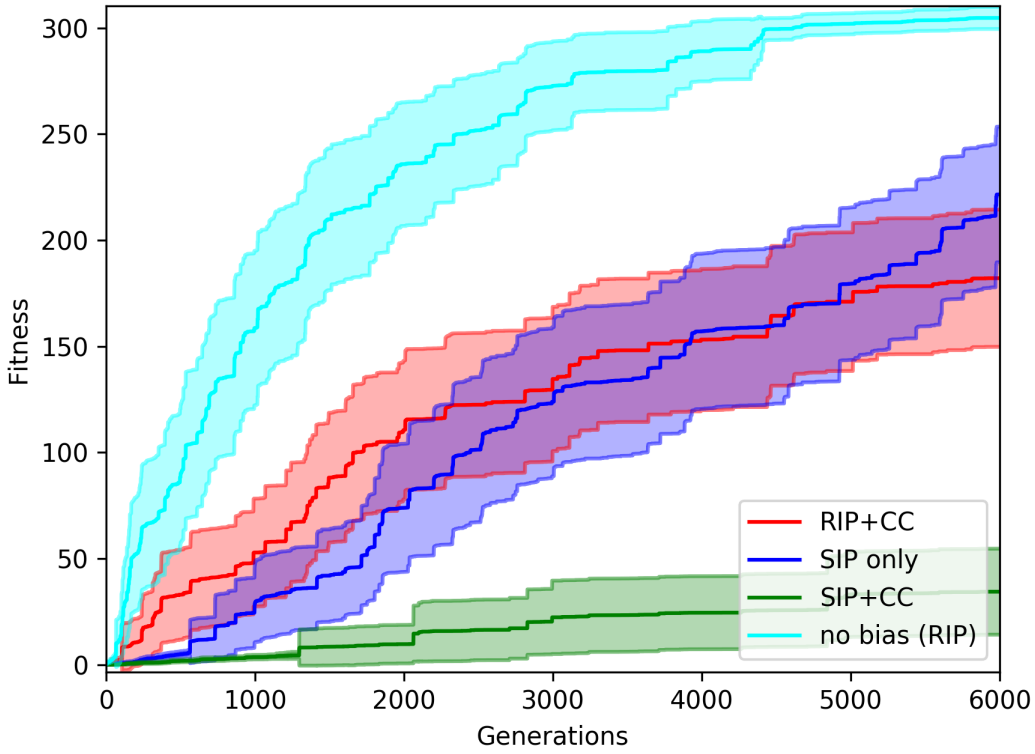


Figure 4.4: Performance of the scaffolding fitness function evolution under various forms of bias towards sparsity: (red line) with initial population of random networks and connection cost co-optimization - RIP-CC; (green line) with initial population of sparse networks and connection cost co-optimization - SIP-CC; (blue line) with initial population of sparse networks only - SIP; (cyan line) with no bias. Error strips show 95% confidence interval of Student's t-distribution based on a sample of 50 runs.

Lastly, we investigate the performance of our approach under a bias towards sparsity (figure 4.4). We apply three methods of such biasing – initialization with populations of sparse networks (section 4.3.3), biasing of the selection algorithm itself (section 4.3.4) and the combination of the two. In all of these cases the algorithm converged less rapidly than in the case when the bias was disabled.

4.5 Discussion and conclusion

We have developed a metaevolutionary algorithm for co-optimizing scaffolding and solution candidates. Despite using less evaluations to optimize each scaffolding function, the algorithm outperforms both manually designed and randomly generated scaffolding fitness functions. The algorithm is applicable to a wide range of design problems in which "natural" fitness functions exhibit the bootstrap problem, yet useful building blocks are available to the designers in form of auxiliary fitness functions.

We have shown that, in line with the predictions of the reformulation framework, biasing the underlying search towards sparsity decreases the convergence rate of the co-optimization algorithm. This is in contrast to the case of co-optimization of morphology and control, where biasing control evolution towards sparse solution candidates can increase the convergence rate [6] (*note - also provided as Chapter 3*).

References for chapter 4

[5] Anton Bernatskiy and Josh Bongard. "Choice of robot morphology can prohibit modular control and disrupt evolution". In: *Proceedings of the 14th European Conference on Artificial Life*. MIT. 2017, pp. 60–67.

[6] Anton Bernatskiy and Josh Bongard. "Evolving morphology automatically reformulates the problem of designing modular control". In: *Adaptive Behavior* 26.2 (2018), pp. 47–64.

[7] Anton Bernatskiy and Joshua C Bongard. "Exploiting the relationship between structural modularity and sparsity for faster network evolution". In: *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM. 2015, pp. 1173–1176.

[11] Josh Bongard. "Behavior Chaining-Incremental Behavior Integration for Evolutionary Robotics." In: *ALIFE*. 2008, pp. 64–71.

[26] Jeff Clune, Jean-Baptiste Mouret, and Hod Lipson. "The evolutionary origins of modularity". In: *Proc. R. Soc. B*. Vol. 280. 1755. The Royal Society. 2013, p. 20122863.

[47] Faustino Gomez and Risto Miikkulainen. "Incremental evolution of complex general behavior". In: *Adaptive Behavior* 5.3-4 (1997), pp. 317–342.

[50] Thomas Helmuth, Lee Spector, and James Matheson. "Solving uncompromising problems with lexicase selection". In: *IEEE Transactions on Evolutionary Computation* 19.5 (2015), pp. 630–643.

[51] Geoffrey E Hinton and Steven J Nowlan. “How learning can guide evolution”. In: *Complex systems* 1.3 (1987), pp. 495–502.

[53] Joost Huizinga and Jeff Clune. “Evolving Multimodal Robot Behavior via Many Stepping Stones with the Combinatorial Multi-Objective Evolutionary Algorithm”. In: *arXiv preprint arXiv:1807.03392* (2018).

[62] Sam Kriegman, Nick Cheney, and Josh Bongard. “How morphological development can guide evolution”. In: *arXiv preprint arXiv:1711.07387* (2017).

[70] Joel Lehman and Kenneth O Stanley. “Abandoning objectives: Evolution through the search for novelty alone”. In: *Evolutionary computation* 19.2 (2011), pp. 189–223.

[73] Xun Li and Risto Miikkulainen. “Evolving multimodal behavior through subtask and switch neural networks”. In: *Proceedings of the Fourteenth International Conference on the Synthesis and Simulation of Living Systems*. Citeseer. 2014.

[76] Hod Lipson et al. “On the origin of modular variation”. In: *Evolution* 56.8 (2002), pp. 1549–1556.

[83] Jean-Baptiste Mouret and Stéphane Doncieux. “Incremental evolution of animats’ behaviors as a multi-objective optimization”. In: *International Conference on Simulation of Adaptive Behavior*. Springer. 2008, pp. 210–219.

[89] Stefano Nolfi and Domenico Parisi. “Evolving non-trivial behaviors on real robots: an autonomous robot that picks up objects”. In: *Congress of the Italian Association for Artificial Intelligence*. Springer. 1995, pp. 243–254.

[102] Michael Schmidt and Hod Lipson. “Age-fitness pareto optimization”. In: *Genetic Programming Theory and Practice VIII*. Springer, 2011, pp. 129–146.

[104] Fernando Silva et al. “Open issues in evolutionary robotics”. In: *Evolutionary computation* 24.2 (2016), pp. 205–236.

[106] Andrew C Slocum, Douglas C Downey, and Randall D Beer. “Further experiments in the evolution of minimally cognitive behavior: From perceiving affordances to selective attention”. In: *From animals to animats 6: Proceedings of the sixth international conference on simulation of adaptive behavior*. 2000, pp. 430–439.

[128] David H Wolpert and William G Macready. “No free lunch theorems for optimization”. In: *IEEE transactions on evolutionary computation* 1.1 (1997), pp. 67–82.

Chapter 5

Conclusion

In this thesis I have shown that in control tasks the morphology of the robot's body can determine if certain tasks can be solved by modular controllers. I demonstrated that, by using evolutionary algorithms biased towards modularity and for morphologies that admit modular control it is possible to produce controllers that solve the task much more rapidly than in the case when the morphology does not admit modular control (chapter 2). I have shown that if the morphology and control are co-optimized by an evolutionary algorithm and the evolution of control is biased towards modularity, then morphological evolution finds the morphologies that admit modular control. Evolution in this case finds more fit solutions more rapidly compared to the case when the morphology is not evolved. I have shown that for this type of setup there is an optimal ratio of mutation rates for morphology and control (chapter 3).

I generalized these results within reformulation approach: a general framework that can be used to describe and reason about optimization systems with multiple time scales. Within this framework I described a new design principle of guided

reformulation (chapter 3).

The next section of this chapter describes the reformulation approach more formally than chapter 3, giving a quantitative interpretation to the guided reformulation heuristic and to a certain type of deception that occurs as an artifact of multiple optimization time scales. In section 5.2 I discuss broader impact of my results. Finally, in section 5.3 I discuss some possible extensions of the present work.

5.1 Formal Theory of Reformulation

Reformulation approach is based on the theory of insight through reformulation [25, 127] from cognitive science. The way I interpret this theory is as follows: cognitive insight or epiphany may occur as a result of searching through the space of the approaches to the problem until the process of solving the problem becomes trivial.

In stacked optimization this idea translates into searching the space of extra parameters of optimization or design (driving variables) to minimize the difficulty of optimizing the primary solution parameters (non-driving variables) (see chapter 3). In this section I describe this idea more formally and give quantitative explanations to the principle of guided reformulation and to the problem of premature convergence that can be induced by the reformulation approach on certain landscapes.

Consider a task of finding a maximum of a parametric function $f_a(x) : X_a \to \mathbb{R}$, $a \in A$. The maximum is searched for over the full domain of the function $\{x, a | a \in A, x \in X(a)\}$. For example, f might be a fitness function of a robot, the value that depends on its morphology $a \in A$ and its controller $x \in X(a)$. Since the number of sensors and motors depends on the morphology, the set of all possible controllers

$X(a)$ depends on the morphology.

Suppose that there is a family of optimization methods $\mathcal{P}_b$, $b \in B$ that can find a maximum f_a^{max} of f_a for any $a \in A$. Here, $b \in B$ stands for the parameters of optimization method that are available to the designer. For example, if $\mathcal{P}$ is a family of evolutionary algorithms, then b might include mutation rate, population size etc.

In the absence of additional constraints, the solution of this problem is straightforward: the designer chooses a reasonable value of b (possibly dependent on a) and applies $\mathcal{P}_b$ to find f_a^{max} of $f_a(x)$ over $X(a)$. A search of a maximum of f_a^{max} over the values of a is then performed with any optimization algorithm, e.g. by exhaustively enumerating A.

In practice, however, resource constraints often make this approach impractical. If no additional information about the function f_a is available, then finding the exact value of f_a^{max} will require exhaustively searching the function's domain $X(a)$. For practical problems this is often a prohibitively expensive operation. Instead of using this "brute force" strategy, iterative algorithms are employed that provide an estimation of f_a^{max} that may get better as more resources are spent on the optimization.

I begin by considering the case of deterministic optimization algorithms. To this end I introduce **effort function** $\tau_{a,b}(y) : f_a(X_a) \to \mathbb{R}_{>0}$ that describes the amount of resources (e.g. computational) that is required to obtain the estimate y of f_a^{max} using the optimization system $\mathcal{P}_b$. Since any estimate that can be obtained using a certain amount of resources can always be obtained using a larger amount, this function is monotonically increasing. Note that the effort function depends both on the parameters of the task a and on the parameters of the optimization system b.

An example of an effort function would be the number of gradient ascent steps

that is required to get an estimate y of the maximum value of the function $-\alpha x^2$. The parameters in this case are the coefficient α ($a = [\alpha]$), starting position x_0 and the learning rate γ ($b = [x_0, \gamma]$).

The new task is to obtain the biggest possible value of the maximum estimate given a certain fixed resource budget T. In certain cases (e.g. in classical control optimization) it is possible to choose reasonable values of a and b based on the available domain knowledge and get "good-enough" results by optimizing f with these fixed values of the parameters.

In other cases a certain small (much less than T) amount of resources spent on optimization for any $a \in A$ and some b ensures that the value produced by the optimizer is near the value that corresponds to infinite resources. This enables designers to focus on the quality of final solutions instead of the optimization rate when choosing parameter values. A common example of this type of metaoptimization is hyperparameter tuning in machine learning.

Here we will focus on the most difficult case: there is no direct knowledge of the effort function and the effort required to estimate the final value produced by the optimization process is, for most sets of parameters, comparable to the budget T. In this case it is crucial to optimize the parameters to make the convergence of $\mathcal{P}$ more rapid. Due to the resemblance that such approaches bear to one hypothetic mechanism of cognitive insight [127] I call this methodology **reformulation** in optimization.

Reformulation divides the resources between several optimization processes with different values of the parameters. The simplest strategy is to allocate a small amount of resources to each $(a, b) \in A \times B$, measure the increase in fitness that resulted in each case and allocate the rest of the resources to the value that produced the best

fitness (fig. 5.1). This approach works well if the effort function is such that for every value of the parameters the results that are obtained with any fixed amount of effort are indicative of the results that can be obtained with any larger amount of effort (fig. 5.1 (a)). I will call effort functions that possess this property **convex**. For non-convex effort functions this simple strategy may yield inconclusive results (fig. 5.1 (b)) or converge prematurely to the parameter values that permit more rapid improvement of the solution, but ultimately reach lower fitness (fig. 5.1 (c)).

Often, size of the set $A \times B$ prevents all the values of the parameters to be explored at once. Typical strategy in this case is to explore a subset of the parameters settings at any given time. It is important, however, to compare parameters setting using the same amount of resources. If the resources are allocated in parallel to several competing parameters settings, then the ones that are considered for a longer period of time will reach higher values of fitness due to more resources spent and outcompete even the parameters settings that have superior rate of convergence. Thus, in memoryless systems with competition even convex effort functions are deceptive! This problem is at heart of the difficulty of co-evolving morphology and modularity [20].

Innovation protection through Pareto optimization [102, 22] is an efficient way to cope with both non-convexity of the effort function and with the deception due to parallel resource allocation. In this approach, parameter settings that received less resources can, if they yield higher fitness, displace the parameter settings that received more, but not the other way around. As the total allocated amount of resources T grows, resources allocated for any given parameters setting increase indefinitely, and in the limit of $T \to \infty$ the method is guaranteed to find the parameters setting that produces the best result with this amount.

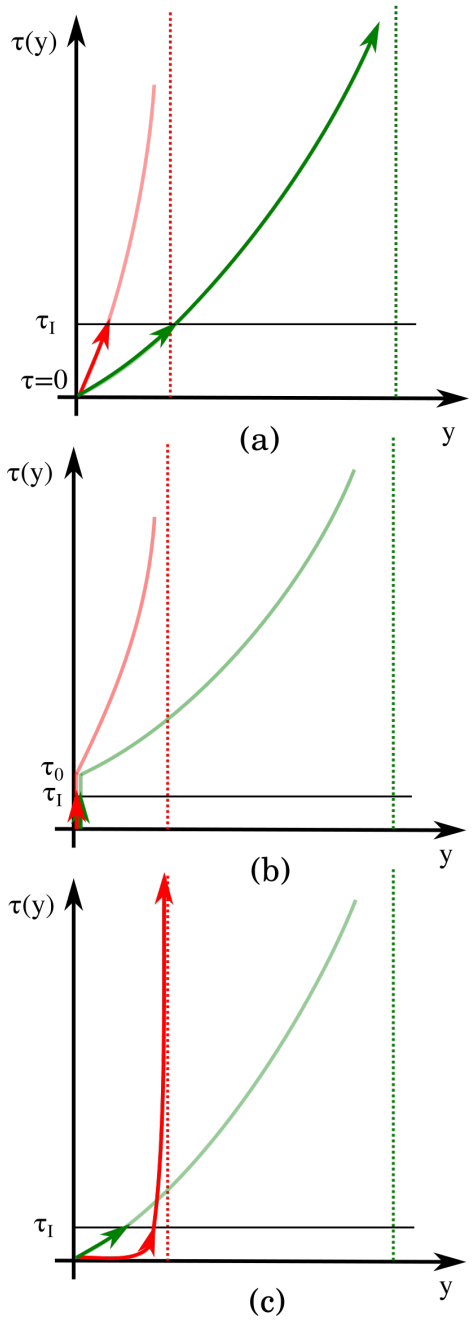


Figure 5.1: Simple reformulation strategy and its failure modes. In this approach, each settings of parameters is evaluated with a certain small amount of resources $\tau_I \ll T$. The approach that yields the best fitness gets the bulk of the resources. (a) For a convex effort function the approach succeeds, i.e. it finds the setting of the parameters that maximizes the convergence rate and the final result. (b) If τ_I is insufficient to improve fitness for any setting of the parameters, there is no winner. (c) If τ_I is insufficient to discriminate between an approach that converges rapidly, but to a low value of fitness, and an approach that converges slowly, but to a higher value, then the wrong combination of parameters will be chosen.

Another issue of the reformulation methodology has to do with the fact that not all parameters in a and b have appreciable impact on how rapidly the convergence occurs. If the parameters that are varied during the optimization do not affect the rate of convergence, reformulation approaches will not be much more effective than simply using all the budget on a single value of the parameters selected randomly. Hence, it is crucial to ensure that at least some parameters do influence the convergence rate. Following the terminology of chapter 3, I'll call them **driving variables**.

Presence of strong effect of some variables in a and b on the convergence rate becomes even more important once we consider a more realistic setting in which the optimization algorithm is stochastic. In this case effort function becomes **effort distribution function** which maps the effort spent optimizing f_a with $\mathcal{P}_b$ to a probability distribution of the outcome (i.e., the maximum estimate produced). If variance of such distribution is comparable to the size of the effect of changing a variable, then parameters optimization algorithm will need many costly evaluations to determine the effect of changing the parameters. On the other hand, if the variable is pronouncedly driving (i.e. has a strong effect on how fast the algorithm converges, compared to variances of the outcome distributions), then a single short run of the optimization algorithm will show which value of the variable leads to a more rapid convergence with high probability.

Certain parameters can influence how other parameters affect the convergence rate of the underlying search. By choosing values of such parameters it is possible to make driving strongly variables out of variables that have slight or no effect on the convergence rate otherwise. This observation is at the heart of the **guided reformulation** heuristic (chapter 3) which states that whether or not a bias is applied

to the underlying optimization process is one such variable. The heuristic itself is as follows: if domain knowledge available suggests that for a certain value of a certain task parameter a_i or optimization process parameter b_i some bias makes the task very simple, then the reformulation process is likely to be more rapid if the bias is applied.

5.2 Significance Of Findings

In the opinion of the author, the biggest contribution of the present thesis is the reformulation framework. Although the idea of reformulation is not novel (see e.g. [25]), to the best of my knowledge, this thesis is the first instance of this idea being used to tie multiple concepts from the field of global optimization into a coherent picture. The result enables to reason and make predictions about optimization processes with multiple time scales, as I've done in chapters 3 and 4.

One idea developed within this framework is the guided reformulation heuristic that can make it possible to automatically reformulate and solve some tasks that otherwise resist the reformulation approach. I applied it to the synthetic task of optimizing Arrowbot controllers (chapter 3), but it can be applied to any task for which there is a bias in the primary variables optimization that is known to trivialize the task for some unknown values of some additional variables.

Author hopes that these concepts find applications in the development of optimizing systems and evolutionary theory. The ideas are especially relevant to evolutionary robotics and reinforcement learning, although their scope is by no means limited to these areas.

In addition, several result specific to robotics were obtained. I have shown that

morphology can influence feasibility of modular control, both analytically and by evolving the controllers (chapter 2). This provides a useful heuristic for robot designers: modularity of control must be considered at the same stage as when robot's body is developed. Analytical side of my work provides some insights into the interplay between the sensorymotor interactions induced by the body and modular control, which again may be useful in robotic design.

Further, I have shown that the morphology-modularity dependence can be used to systematically search for robots that can be controlled with simple, modular controllers (chapter 3). Although the exact scope of this method remains unknown, it has shown promising results for at least one task and environment.

Finally, I developed a new method for evolving modular networks by initializing search with sparse networks (Appendix A.1). The method does not require adding any Pareto optimization objectives, which makes it ideal for the cases when number of such objectives is a concern. Simplicity of the method allows it to be used in a wide variety of network optimization applications.

To conclude, this dissertation tackles two out of three issues I listed that contribute to preventing evolutionary robotics and design-by-optimization approaches to engineering from scaling up: catastrophic forgetting and the bootstrap problem. Additionally, deception is studied to some extent in chapter 3. Although it solves none of these issues, it provides a toolset – reformulation theory – for attacking at least some of their aspects. Author hopes that the contributions will be useful in the development of autonomous agents that can operate in complex environments.

5.3 Future Work

More information on the scope of reformulation approach is needed, especially in its relation to modularity in evolutionary robotics:

1. There seems to be a deep connection between the structure of the environment, agent's morphology and feasibility/evolvability of modular control. Research question that can be asked here is as follows: **is it possible to make an *a priori*, heuristic guess about which design variables are likely to influence the optimization rate of others strongly and under which conditions?** Within this work I established that morphology is one such group of variables if control is optimized with a bias towards modularity. I do hope, however, that it is possible to be more specific and find out which features of morphology make its influence on modularity of control so profound.

   An answer to this question would inevitably involve some general model of the task-environment-robot's body and the morphological degrees of freedom. Ashby [1] used diagrams of immediate effects, but my preliminary investigations show that these are not sufficient in some practically important settings (e.g. locomotion). Different types of causal or dynamical will have to be considered.

2. A related problem is defining the mutation operator for the slow timescale in such a way that it is not likely to change the rate of convergence of the fast timescale by much. If the mutation operator does not satisfy this requirement, the slow timescale part of the evolution is likely to only be as efficient as random search. With the hypothesized direct relation between the existence on a

modular solution and the rate of convergence of the fast timescale, the question can be reformulated as follows: **given a choice of driving variables, it is possible to design a way to mutate them that is unlikely to not change the convergence rate of the fast timescale by much?**

Again, solving this task requires some a general world model to figure our how a mutation operator tweaking the driving variables might influence the structure of the solutions.

3. One particularly prospective idea for searching for strongly driving variables that rely on the modularity bias is the idea of **conditional modularity**. In many complex systems, keeping certain variables within certain limits drastically reduces the complexity of the interactions within the systems and allows for simpler control. Some examples include the constant body temperature in birds and mammals that makes it possible to exhibit a richer set of biochemical behaviors; in metal recycling, heating the scrap above its melting point helps to avoid dealing with its complex structure. In both of these examples temperature happens to be the variable that governs the complexity of the system. **How can we find such variables given some limited information about the class of system and possibly the capacity to run some experiments?**

4. Preliminary investigations suggest that in some tasks, **time pressure can be the factor that pushes the optimal solutions towards more dense dependence of parts**. Since this factor is present in many practically important tasks, it is desirable to investigate its interaction with the evolutionary techniques biased towards structural modularity (and thus sparsity).

5. To increase the range of the systems for which the approach is helpful, **the interaction of the approach with generative encodings** for morphology and control should be considered.

Bibliography

[1] W Ross Ashby. *Design for a Brain: The origin of adaptive behavior*. 2nd. London: Chapman & Hall Ltd, 1960.

[2] Robert Axelrod et al. "The evolution of strategies in the iterated prisoner's dilemma". In: *The dynamics of norms* (1987), pp. 1–16.

[3] Carliss Y Baldwin and Kim B Clark. "Modularity in the design of complex engineering systems". In: *Complex engineered systems*. Springer, 2006, pp. 175–205.

[4] Andrea Banino et al. "Vector-based navigation using grid-like representations in artificial agents". In: *Nature* 557.7705 (2018), p. 429.

[5] Anton Bernatskiy and Josh Bongard. "Choice of robot morphology can prohibit modular control and disrupt evolution". In: *Proceedings of the 14th European Conference on Artificial Life*. MIT. 2017, pp. 60–67.

[6] Anton Bernatskiy and Josh Bongard. "Evolving morphology automatically reformulates the problem of designing modular control". In: *Adaptive Behavior* 26.2 (2018), pp. 47–64.

[7] Anton Bernatskiy and Joshua C Bongard. "Exploiting the relationship between structural modularity and sparsity for faster network evolution". In: *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM. 2015, pp. 1173–1176.

[8] Anton Bernatskiy, Gregory S Hornby, and Josh C Bongard. "Improving robot behavior optimization by combining user preferences". In: *Artificial Life* 14 (2014).

[9] Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[10] Vincent D Blondel et al. "Fast unfolding of communities in large networks". In: *Journal of Statistical Mechanics: Theory and Experiment* 2008.10 (2008), P10008.

[11] Josh Bongard. “Behavior Chaining-Incremental Behavior Integration for Evolutionary Robotics.” In: *ALIFE*. 2008, pp. 64–71.

[12] Josh C Bongard. “Spontaneous evolution of structural modularity in robot neural network controllers”. In: *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. ACM. 2011, pp. 251–258.

[13] Josh C Bongard and Gregory S Hornby. “Combining fitness-based search and user modeling in evolutionary robotics”. In: *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. ACM. 2013, pp. 159–166.

[14] Josh C Bongard et al. “Evolving robot morphology facilitates the evolution of neural modularity and evolvability”. In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM. 2015, pp. 129–136.

[15] Josh Bongard, Victor Zykov, and Hod Lipson. “Resilient machines through continuous self-modeling”. In: *Science* 314.5802 (2006), pp. 1118–1121.

[16] Rodney A Brooks. “Intelligence without representation”. In: *Artificial intelligence* 47.1 (1991), pp. 139–159.

[17] Brett Calcott. “Chaining Distinct Tasks Drives The Evolution of Modularity”. In: *ALIFE 14: The Fourteenth Conference on the Synthesis and Simulation of Living Systems*. 2014, pp. 701–702.

[18] Collin K Cappelle et al. “Morphological Modularity Can Enable the Evolution of Robot Behavior to Scale Linearly with the Number of Environmental Features”. In: *Frontiers in Robotics and AI* 3 (2016), p. 59.

[19] Sean B Carroll. “Chance and necessity: the evolution of morphological complexity and diversity”. In: *Nature* 409.6823 (2001), pp. 1102–1109.

[20] Nick Cheney et al. “On the Difficulty of Co-Optimizing Morphology and Control in Evolved Virtual Creatures”. In: *Proceedings of Artificial Life Conference*. 2016, pp. 226–234.

[21] Nick Cheney et al. “Scalable Co-Optimization of Morphology and Control in Embodied Machines”. In: *arXiv preprint arXiv:1706.06133* (2017).

[22] Nick Cheney et al. “Scalable co-optimization of morphology and control in embodied machines”. In: *Journal of The Royal Society Interface* 15.143 (2018), p. 20170937.

[23] Nick Cheney et al. “Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding”. In: *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. ACM. 2013, pp. 167–174.

[24] Andrea Cherubini et al. “Collaborative manufacturing with physical human–robot interaction”. In: *Robotics and Computer-Integrated Manufacturing* 40 (2016), pp. 1–13.

[25] Berthe Y Choueiry, Yumi Iwasaki, and Sheila McIlraith. “Towards a practical theory of reformulation for reasoning about physical systems”. In: *Artificial Intelligence* 162.1-2 (2005), pp. 145–204.

[26] Jeff Clune, Jean-Baptiste Mouret, and Hod Lipson. “The evolutionary origins of modularity”. In: *Proc. R. Soc. B*. Vol. 280. 1755. The Royal Society. 2013, p. 20122863.

[27] Ian A Crawford. “Dispelling the myth of robotic efficiency”. In: *Astronomy & Geophysics* 53.2 (2012), pp. 2–22.

[28] Antoine Cully et al. “Robots that can adapt like animals”. In: *Nature* 521.7553 (2015), pp. 503–507.

[29] Yann N Dauphin et al. “Identifying and attacking the saddle point problem in high-dimensional non-convex optimization”. In: *Advances in neural information processing systems*. 2014, pp. 2933–2941.

[30] Ltsc Deng et al. “Recent advances in deep learning for speech research at Microsoft.” In: *ICASSP*. Vol. 26. 2013, p. 64.

[31] Jeremy Draghi and Günter P Wagner. “Evolution of evolvability in a developmental model”. In: *Evolution* 62.2 (2008), pp. 301–315.

[32] Karl Duncker and Lynne S Lees. “On problem-solving.” In: *Psychological monographs* 58.5 (1945), p. i.

[33] Peter Durr, Dario Floreano, and Claudio Mattiussi. “Genetic representation and evolvability of modular neural controllers”. In: *IEEE Computational Intelligence Magazine* 5.3 (2010), pp. 10–19.

[34] Kai Olav Ellefsen, Jean-Baptiste Mouret, and Jeff Clune. “Neural modularity helps organisms evolve to learn new skills without forgetting old skills”. In: *PLoS Comput Biol* 11.4 (2015), e1004128.

[35] Jacob A Englander and Bruce A Conway. “Automated Solution of the Low-Thrust Interplanetary Trajectory Problem”. In: *Journal of Guidance, Control, and Dynamics* (2016), pp. 15–27.

[36] Carlos Espinosa-Soto and Andreas Wagner. “Specialization can drive the evolution of modularity”. In: *PLoS Comput Biol* 6.3 (2010), e1000719.

[37] Jalal Etesami and Negar Kiyavash. “Measuring Causal Relationships in Dynamical Systems through Recovery of Functional Dependencies”. In: *IEEE Transactions on Signal and Information Processing over Networks* (2016).

[38] Adrienne L Fairhall et al. “Multiple timescales of adaptation in a neural code”. In: *Advances in neural information processing systems*. 2001, pp. 124–130.

[39] Brian S Fisher and Sabine Schnittger. “Autonomous and remote operation technologies in the mining industry”. In: *BAeconomics Pty Ltd, February* (2012).

[40] Dario Floreano and Francesco Mondada. “Automatic creation of an autonomous agent: Genetic evolution of a neural network driven robot”. In: *Proceedings of the third international conference on Simulation of adaptive behavior: From Animals to Animats 3*. LIS-CONF-1994-003. MIT Press. 1994, pp. 421–430.

[41] Robert A Freitas Jr and Ralph C Merkle. *Kinematic self-replicating machines*. Landes Bioscience, 2004.

[42] Robert A Freitas and William P Gilbreath. “Advanced automation for space missions”. In: *Journal of the Astronautical Sciences* 30 (1982), pp. 1–11.

[43] Robert M French. “Catastrophic forgetting in connectionist networks”. In: *Trends in cognitive sciences* 3.4 (1999), pp. 128–135.

[44] Thomas Geijtenbeek, Michiel van de Panne, and A. Frank van der Stappen. “Flexible Muscle-Based Locomotion for Bipedal Creatures”. In: *ACM Transactions on Graphics* 32.6 (2013).

[45] Michelle Girvan and Mark EJ Newman. “Community structure in social and biological networks”. In: *Proceedings of the national academy of sciences* 99.12 (2002), pp. 7821–7826.

[46] David E Goldberg, Jon Richardson, et al. “Genetic algorithms with sharing for multimodal function optimization”. In: *Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms*. Hillsdale, NJ: Lawrence Erlbaum. 1987, pp. 41–49.

[47] Faustino Gomez and Risto Miikkulainen. “Incremental evolution of complex general behavior”. In: *Adaptive Behavior* 5.3-4 (1997), pp. 317–342.

[48] Joel Hasbrouck and Gideon Saar. “Low-latency trading”. In: *Journal of Financial Markets* 16.4 (2013), pp. 646–679.

[49] Jeff Hawkins and Sandra Blakeslee. *On intelligence: How a new understanding of the brain will lead to the creation of truly intelligent machines*. Macmillan, 2007.

[50] Thomas Helmuth, Lee Spector, and James Matheson. “Solving uncompromising problems with lexicase selection”. In: *IEEE Transactions on Evolutionary Computation* 19.5 (2015), pp. 630–643.

[51] Geoffrey E Hinton and Steven J Nowlan. "How learning can guide evolution". In: *Complex systems* 1.3 (1987), pp. 495–502.

[52] Gregory Hornby et al. "Automated antenna design with evolutionary algorithms". In: *Space 2006*. 2015, p. 7242.

[53] Joost Huizinga and Jeff Clune. "Evolving Multimodal Robot Behavior via Many Stepping Stones with the Combinatorial Multi-Objective Evolutionary Algorithm". In: *arXiv preprint arXiv:1807.03392* (2018).

[54] Julian Huxley. *Evolution the modern synthesis*. George Allen and Unwin, 1942.

[55] Fumiya Iida, Raja Dravid, and Chandana Paul. "Design and control of a pendulum driven hopping robot". In: *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*. Vol. 3. IEEE. 2002, pp. 2141–2146.

[56] Nadav Kashtan and Uri Alon. "Spontaneous evolution of modularity and network motifs". In: *Proceedings of the National Academy of Sciences of the United States of America* 102.39 (2005), pp. 13773–13778.

[57] Stefan J Kiebel et al. "Recognizing sequences of sequences". In: *PLoS computational biology* 5.8 (2009), e1000464.

[58] Jon Kleinberg and Eva Tardos. *Algorithm design*. Pearson, 2005.

[59] Günther Knoblich, Stellan Ohlsson, and Gary E Raney. "An eye movement study of insight problem solving". In: *Memory & cognition* 29.7 (2001), pp. 1000–1009.

[60] Loizos Kounios et al. "Resolving the paradox of evolvability with learning theory: How evolution learns to improve evolvability on rugged fitness landscapes". In: *arXiv preprint arXiv:1612.05955* (2016).

[61] Kostas Kouvaris et al. "How evolution learns to generalise: Using the principles of learning theory to understand the evolution of developmental organisation". In: *PLOS Computational Biology* 13.4 (2017), e1005358.

[62] Sam Kriegman, Nick Cheney, and Josh Bongard. "How morphological development can guide evolution". In: *arXiv preprint arXiv:1711.07387* (2017).

[63] Sam Kriegman et al. "Evolving Spatially Aggregated Features from Satellite Imagery for Regional Modeling". In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2016, pp. 707–716.

[64] Scott Kuindersma et al. "Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot". In: *Autonomous Robots* 40.3 (2016), pp. 429–455.

[65] Tin Lun Lam and Yangsheng Xu. "Biologically inspired tree-climbing robot with continuum maneuvering mechanism". In: *Journal of Field Robotics* 29.6 (2012), pp. 843–860.

[66] Christopher G Langton. *Artificial life: An overview*. Mit Press, 1997.

[67] Richard Larn and Rex Whistler. *Commercial diving manual*. David & Charles, 1993.

[68] Ari Larson et al. "Recombination Hotspots Promote the Evolvability of Modular Systems". In: *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*. ACM. 2016, pp. 115–116.

[69] Joel Lehman et al. "The surprising creativity of digital evolution: A collection of anecdotes from the evolutionary computation and artificial life research communities". In: *arXiv preprint arXiv:1803.03453* (2018).

[70] Joel Lehman and Kenneth O Stanley. "Abandoning objectives: Evolution through the search for novelty alone". In: *Evolutionary computation* 19.2 (2011), pp. 189–223.

[71] Joel Lehman and Kenneth O Stanley. "Exploiting Open-Endedness to Solve Problems Through the Search for Novelty." In: *ALIFE*. 2008, pp. 329–336.

[72] Jiwei Li et al. "Deep reinforcement learning for dialogue generation". In: *arXiv preprint arXiv:1606.01541* (2016).

[73] Xun Li and Risto Miikkulainen. "Evolving multimodal behavior through subtask and switch neural networks". In: *Proceedings of the Fourteenth International Conference on the Synthesis and Simulation of Living Systems*. Citeseer. 2014.

[74] Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[75] Hod Lipson and Jordan B Pollack. "Automatic design and manufacture of robotic lifeforms". In: *Nature* 406.6799 (2000), pp. 974–978.

[76] Hod Lipson et al. "On the origin of modular variation". In: *Evolution* 56.8 (2002), pp. 1549–1556.

[77] Paul Martin, Paul Patrick Gordon Bateson, and Patrick Bateson. *Measuring behaviour: an introductory guide*. Cambridge University Press, 1993.

[78] Maja J Matarić. "Behavior-based robotics as a tool for synthesis of artificial behavior and analysis of natural behavior". In: *Trends in cognitive sciences* 2.3 (1998), pp. 82–86.

[79] Maja Matarić and Dave Cliff. "Challenges in evolving controllers for physical robots". In: *Robotics and autonomous systems* 19.1 (1996), pp. 67–83.

[80] Risto Miikkulainen et al. “Evolving Deep Neural Networks”. In: *arXiv preprint arXiv:1703.00548* (2017).

[81] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), pp. 529–533.

[82] Jean-Baptiste Mouret. “Novelty-based multiobjectivization”. In: *New horizons in evolutionary robotics*. Springer, 2011, pp. 139–154.

[83] Jean-Baptiste Mouret and Stéphane Doncieux. “Incremental evolution of animats’ behaviors as a multi-objective optimization”. In: *International Conference on Simulation of Adaptive Behavior*. Springer. 2008, pp. 210–219.

[84] Jun Nakanishi, Toshio Fukuda, and Daniel E Koditschek. “A brachiating robot controller”. In: *IEEE Transactions on Robotics and Automation* 16.2 (2000), pp. 109–123.

[85] Mark EJ Newman. “Modularity and community structure in networks”. In: *Proceedings of the national academy of sciences* 103.23 (2006), pp. 8577–8582.

[86] Anh Mai Nguyen, Jason Yosinski, and Jeff Clune. “Innovation engines: Automated creativity and improved stochastic optimization via deep learning”. In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM. 2015, pp. 959–966.

[87] Stefano Nolfi and Dario Floreano. “Coevolving predator and prey robots: Do "arms races" arise in artificial evolution?” In: *Artificial life* 4.4 (1998), pp. 311–335.

[88] Stefano Nolfi and Dario Floreano. *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*. MIT press, 2000.

[89] Stefano Nolfi and Domenico Parisi. “Evolving non-trivial behaviors on real robots: an autonomous robot that picks up objects”. In: *Congress of the Italian Association for Artificial Intelligence*. Springer. 1995, pp. 243–254.

[90] Steven Orzack, Steven Hecht Orzack, and Elliott Sober. *Adaptationism and optimality*. Cambridge University Press, 2001.

[91] Chandana Paul. “Morphology and computation”. In: *Proceedings of the International Conference on the Simulation of Adaptive Behaviour Los Angeles, CA, USA*. 2004, pp. 33–38.

[92] Rolf Pfeifer and Josh Bongard. *How the body shapes the way we think: a new view of intelligence*. MIT press, 2006.

[93] Rolf Pfeifer, Fumiya Iida, and Gabriel Gómez. “Morphological computation for adaptive behavior and cognition”. In: *International Congress Series*. Vol. 1291. Elsevier. 2006, pp. 22–29.

[94] Kenneth Price, Rainer M Storn, and Jouni A Lampinen. *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media, 2006.

[95] Marc Raibert et al. "Bigdog, the rough-terrain quadruped robot". In: *IFAC Proceedings Volumes* 41.2 (2008), pp. 10822–10825.

[96] Torsten Reil and Phil Husbands. "Evolution of central pattern generators for bipedal walking in a real-time physics environment". In: *IEEE Transactions on Evolutionary Computation* 6.2 (2002), pp. 159–168.

[97] Marieke Rohde. *Enaction, embodiment, evolutionary robotics: simulation models for a post-cognitivist science of mind*. Vol. 1. Springer Science & Business Media, 2010.

[98] Christopher D Rosin and Richard K Belew. "New methods for competitive coevolution". In: *Evolutionary computation* 5.1 (1997), pp. 1–29.

[99] Leonel Rozo et al. "Learning physical collaborative robot behaviors from human demonstrations". In: *IEEE Transactions on Robotics* 32.3 (2016), pp. 513–527.

[100] Stuart Jonathan Russell and Peter Norvig. *Artificial intelligence: a modern approach*. 3rd ed. Prentice Hall, 2009.

[101] Martin DJ Sayer. "NOAA Diving Manual: Diving for Science and Technology". In: *Underwater Technology* 31.4 (2013), pp. 217–218.

[102] Michael Schmidt and Hod Lipson. "Age-fitness pareto optimization". In: *Genetic Programming Theory and Practice VIII*. Springer, 2011, pp. 129–146.

[103] Fernando Silva, Luís Correia, and Anders Lyhne Christensen. "A case study on the scalability of online evolution of robotic controllers". In: *Portuguese Conference on Artificial Intelligence*. Springer. 2015, pp. 189–200.

[104] Fernando Silva et al. "Open issues in evolutionary robotics". In: *Evolutionary computation* 24.2 (2016), pp. 205–236.

[105] Karl Sims. "Evolving 3D morphology and behavior by competition". In: *Artificial life* 1.4 (1994), pp. 353–372.

[106] Andrew C Slocum, Douglas C Downey, and Randall D Beer. "Further experiments in the evolution of minimally cognitive behavior: From perceiving affordances to selective attention". In: *From animals to animats 6: Proceedings of the sixth international conference on simulation of adaptive behavior*. 2000, pp. 430–439.

[107] Ricard V Solé and Pau Fernández. "Modularity" for free" in genome architecture?" In: *arXiv preprint q-bio/0312032* (2003).

[108] Steven Squyres. *Roving Mars: Spirit, Opportunity, and the exploration of the red planet*. Hachette Books, 2005.

[109] Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. "A hypercube-based encoding for evolving large-scale neural networks". In: *Artificial life* 15.2 (2009), pp. 185–212.

[110] Kenneth O Stanley and Risto Miikkulainen. "A taxonomy for artificial embryogeny". In: *Artificial Life* 9.2 (2003), pp. 93–130.

[111] Kenneth O Stanley and Risto Miikkulainen. "Competitive coevolution through evolutionary complexification". In: *J. Artif. Intell. Res.(JAIR)* 21 (2004), pp. 63–100.

[112] Kenneth O Stanley and Risto Miikkulainen. "Evolving neural networks through augmenting topologies". In: *Evolutionary computation* 10.2 (2002), pp. 99–127.

[113] Robert J Sternberg and Janet E Davidson. *The nature of insight*. The MIT Press, 1995.

[114] Strabo. *Geography*. 1903 translation by Hamilton, H.C. Esq. and Falconer, W. M.A., retrieved from the Perseus Digital Library 2017-10-30, reference Strab. 8.7.1. 23AD.

[115] Nam P Suh. *The principles of design*. 6. Oxford University Press on Demand, 1990.

[116] Jackrit Suthakorn, Andrew B Cushing, and Gregory S Chirikjian. "An autonomous self-replicating robotic system". In: *Proceedings of*. 2003, pp. 137–142.

[117] Niko Tinbergen. "On aims and methods of ethology". In: *Ethology* 20.4 (1963), pp. 410–433.

[118] Elio Tuci, Gianluca Massera, and Stefano Nolfi. "Active categorical perception in an evolved anthropomorphic robotic arm". In: *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*. IEEE. 2009, pp. 31–38.

[119] Nachum Ulanovsky et al. "Multiple time scales of adaptation in auditory cortex neurons". In: *Journal of Neuroscience* 24.46 (2004), pp. 10440–10453.

[120] Conrad H Waddington. "Canalization of development and the inheritance of acquired characters". In: *Nature* 150.3811 (1942), pp. 563–565.

[121] Gunter P Wagner and Lee Altenberg. "Perspective: Complex adaptations and the evolution of evolvability". In: *Evolution* (1996), pp. 967–976.

[122] Günter P Wagner, Mihaela Pavlicev, and James M Cheverud. "The road to modularity". In: *Nature Reviews Genetics* 8.12 (2007), pp. 921–931.

[123] Jinhua Wang and Zenghua Huang. “The Recent Technological Development of Intelligent Mining in China”. In: *Engineering* 3.4 (2017), pp. 439–444.

[124] Barry Wark, Adrienne Fairhall, and Fred Rieke. “Timescales of inference in visual adaptation”. In: *Neuron* 61.5 (2009), pp. 750–761.

[125] Richard A Watson and Eörs Szathmáry. “How can evolution learn?” In: *Trends in ecology & evolution* 31.2 (2016), pp. 147–157.

[126] John J Welch and David Waxman. “Modularity and the cost of complexity”. In: *Evolution* 57.8 (2003), pp. 1723–1734.

[127] Frank W Wicker et al. “Problem-reformulation training and visualization training with insight problems.” In: *Journal of Educational Psychology* 70.3 (1978), p. 372.

[128] David H Wolpert and William G Macready. “No free lunch theorems for optimization”. In: *IEEE transactions on evolutionary computation* 1.1 (1997), pp. 67–82.

[129] Yuichi Yamashita and Jun Tani. “Emergence of functional hierarchy in a multiple timescale neural network model: a humanoid robot experiment”. In: *PLoS computational biology* 4.11 (2008), e1000220.

[130] Victor Zykov et al. “Robotics: Self-reproducing machines”. In: *Nature* 435.7039 (2005), pp. 163–164.

Appendix A

Appendix

A.1 Exploiting The Relationship Between Structural Modularity And Sparsity For Faster Network Evolution

This work has been published as Bernatskiy, Anton, and Bongard, Josh C.. (2015) "Exploiting the relationship between structural modularity and sparsity for faster network evolution." Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation.

Abstract

A network is structurally modular if it can be divided into tightly connected groups which are weakly connected or disconnected from each other. Such networks are known to be capable of fast evolutionary adaptation, which makes modularity a highly desired property of networks involved in evolutionary computation. Modularity is known to correlate positively with another network parameter, sparsity. Based on this relationship, we hypothesize that starting evolution with a population of sparse networks should increase the modularity of the evolved networks. We find that this technique can enhance the performance of an existing technique for modularity evolution, multiobjective performance-connection cost (P&CC) optimization, and enable a multiobjective algorithm which maximizes performance and minimizes time since last mutation to produce modular solutions almost as efficiently as the P&CC optimization does.

Introduction

Many problems in engineering can be reduced to global optimization of a function $f(G)$ mapping a set of networks (possibly directed and/or weighted) $\mathcal{G}$ onto some set of real numbers $\mathcal{R} \subset \mathbb{R}$. Suppose $\mathcal{G}$ is a set of all possible networks with N nodes and up to $\sim N^2$) edges. Further, each edge has one of C possible weights associated with it (absent connections are defined to have a weight of 0). There are on the order of C^{N^2} such networks. In the worst case, solving the optimization problem involves computing $f(G)$ for all $G \in \mathcal{G}$, i.e. an exponential number of operations.

Two approaches are utilized to cope with this complexity. First, instead of look-

ing for the global optimum, a reasonably good solution which can be obtained in a reasonable time is sought. To this end, metaheuristic methods such as evolutionary computation are utilized. The second approach involves constraining or biasing search towards some small subset of $\mathcal{G}$. The subset is selected heuristically in such a way that it is likely to contain or to be close to some reasonably good solution.

Indirect encodings (e.g. [112]) and additional optimization objectives (e.g. [26]) have been developed which bias the search towards small subsets of the search space. These methods perform well on a wide range of tasks.

A lot of attention has been attracted to the subset of networks possessing structural modularity, a widespread property of biological networks [45]. A network is modular if it can be divided into subgraphs with strong connections within them, but with little or no connections between them. It has been shown that many practical tasks have modular solutions which are reasonably good [12, 17, 26, 36, 56]. It was also found that modular solutions evolve more rapidly than their nonmodular counterparts in nonstationary environments [36] and that they generalize better [1, 121].

A variety of techniques for evolving modular networks has been suggested [26, 36, 56, 17]. One trait that many of these techniques share is an explicit [26] or implicit [36] bias of the evolutionary process towards sparse networks.

Clune et al [26] established that modularity evolves if connection cost is minimized while the network performance is maximized in a Pareto front-based multiobjective evolution. They also demonstrated that such an algorithm produces solutions with higher fitness compared to the case when the performance of a network is the only objective, and that this fitness was arrived at in fewer generations. They concluded

that the influence of the connection cost pushed the population towards the region of the search space with sparser networks. Since sparse networks have less connections and, correspondingly, weights to be optimized, they adapt more rapidly than dense networks.

To explain the increase in modularity, they examined the search space and found a negative correlation between network density and modularity both in random and in highly fit networks.

Alternatively, one can think of this in terms of graph sets: the bias towards sparse networks causes search to optimize fitness on the set of sparse networks first, and this set is much smaller than the set of all possible networks. Due to the relationship between modularity and sparsity mentioned above, the set of sparse networks happens to contain many modular networks, which makes this approach even more efficient.

In [26], evolution was initialized with a population of networks generated by assigning weights to all possible connections at random. Such networks are dense. From the optimization point of view, they are probing the unconstrained set of networks, which is inefficient. They are also not likely to be modular. Plots of density versus modularity versus generation provided in [26] show that it takes a number of generations for the population to reach a region of the search space with sparser and more modular networks.

The fact that the population was seeded with networks with $\mathcal{O}(N^2)$ connections leads us to believe that this transient becomes longer for larger networks. Here we show that seeding evolution with a population of sparse networks can remove this transient, which results in more rapid adaptation and increased modularity, especially for larger values of N. We also demonstrate that, given that the evolution

starts with a population of sparse networks, it is possible to replace the connection cost objective with another objective of minimizing the time since the last mutation without significantly affecting the speed of adaptation. This, however, comes with the expense of destabilizing the process of modularity growth, resulting in a decrease of the final modularity metric Q.

Methods

Here we describe the particular network optimization problem chosen for testing our hypothesis and the algorithms involved. All materials for replicating this work are available at `http://git.io/vUmrG`.

Task: We use evolution of attractors in boolean networks as an example problem for our study. The networks and their dynamic are identical to the ones described in [36].

Boolean networks are dynamical systems which are often used as simple models of gene regulatory networks (GRNs) found in biological cells. The state of each gene j is represented by a variable s_j which can be equal to either -1 or 1. Complete state of the network at time t is a vector of states of N individual genes, $\mathbf{s}(t) = (s_1(t), s_2(t), ..., s_N(t))$. It determines the state of the network at the next time step as follows:

$$s_i(t+1) = \sigma\left[\sum_{j=1}^{N} a_{ij} s_j(t)\right]. \tag{A.1}$$

Here, $a_{ij} \in \{-1, 0, 1\}$ is a strength of connection from gene i to gene j. Function $\sigma(x)$ is defined to be -1 if $x < 0$ and $+1$ otherwise.

Such a network has a point attractor (hereafter referred to as just "attractor") at

state $\mathbf{s}'$ if for some set S of two or more initial conditions the state of the network converges over time to $\mathbf{s}'$ and then stops changing altogether.

The task for the evolutionary algorithms described in this work is to find a matrix of connection strength values A which describes a network with a desired attractor.

Fitness: Fitness of a network of N genes is defined against a target attractor $\mathbf{s}'$. For every target attractor $\mathbf{s}'$ we tested whether the network dynamics converges to it if it starts at a close initial state. We began by generating a set S' of N perturbed attractor states, each of which differs from $\mathbf{s}'$ at exactly one gene. Then we carried out the network dynamics (A.1) starting from every state $\mathbf{s}$ in S' for 20 iterations or until convergence to some state $\mathbf{t}$ (potentially different from $\mathbf{s}'$).

The fitness of the network was then computed as

$$f(\mathbf{s}') = 1 - e^{-3g}, \tag{A.2}$$

where

$$g = \frac{1}{N} \sum_{\mathbf{s} \in S'} (1 - D(\mathbf{s})/N)^5, \tag{A.3}$$

where $D(\mathbf{s})$ is the Hamming distance between $\mathbf{s}$ and $\mathbf{t}$ if the convergence did happen and N otherwise.

Note that this fitness function differs from the one used in [36] only in the way in which the set S'_j is generated. The difference was introduced to reduce the time of computation.

Optimization objectives: We compare the performance of a multiobjective evolutionary algorithm under different sets of objectives. For the sake of uniformity we reduce all objectives to a minimization of some function or property of a network.

Three objectives are used throughout the paper:

1. *Performance (P)* objective, implemented as minimization of $-f$, where f is the fitness function.

2. *Connection cost (CC)* objective, implemented as minimization of the total number of connections.

3. *Time-since-mutation (TSM)* objective, which minimizes the number of generations since the last mutation of the network. The term is chosen to contrast with an established "age" objective [102], which minimizes the number of generations since the emergence of network's family tree. The objective was chosen as to test whether diversity-promoting objective can facilitate the evolution of modular networks and because it does not require a metric in the space of possible behaviors (a requirement for using the novelty objective [82]) nor periodic injections of random genomes into the population ([102]).

All objective sets considered in this paper include exactly two objectives.

Evolutionary algorithm: We employ a simple biobjective evolutionary algorithm which relies on the concept of stochastic Pareto dominance. For a pair of networks (A, B) and a pair of minimizable functions (f, g), we determine whether A stochastically dominates B by first generating a uniformly distributed random number $r \in [0, 1)$ and comparing it to a user-defined constant $p \in [0, 1]$. If $r > p$, only the first objective f is taken into account and the dominance is established if $f(A) < f(B)$. If $r \leqslant p$, both objectives are taken into account and A dominates B if either of the following conditions holds:

1. $f(A) < f(B)$ and $g(A) \leqslant g(B)$,

2. $g(A) < g(B)$ and $f(A) \leqslant f(B)$,

3. $f(A) = f(B)$ and $g(A) = g(B)$ and $ID(A) < ID(B)$.

Here, $ID(X)$ refers to the identification number of the network, a value of a global integer which is incremented every time any network in the population is created or mutated, starting from 0. The value is recorded at the moment of the X network's mutation or creation. Thus, $ID(A) < ID(B)$ indicates that the network A was either generated or mutated before network B.

The constant p describes a probability that the second objective is taken into account. If $p = 1$, stochastic Pareto dominance becomes deterministic, making the comparison between the networks simpler. However, it has been shown in [26] that evolution with the objectives of performance and connection cost (P&CC) has the best convergence rate when p is distant from both 0 and 1. Despite the differences in the stochastic Pareto dominance definition and in the selection strategy, we were able to confirm this result in our preliminary trials (data not shown). Hence, we chose not to switch to deterministic Pareto dominance in our comparison of the P&CC approach to other approaches.

The Pareto front is defined as a subset P' of a population P consisting of all elements of P which are not stochastically dominated by any network in P. At every generation increment, the algorithm finds the Pareto front P' in the current population and adds it to the new population. When done, the algorithm selects a network from the Pareto front at random, makes a copy, mutates it and appends the resulting offspring to the new population. This cycle is repeated until the sizes of the populations become equal, at which point the new population replaces the old one.

In all of our experiments the population was composed of 100 networks.

Mutation operator: Mutation operator acts on network's nodes, having a fixed probability of 0.05 to change the set of strengths of incoming connections to any given gene. One of the following operations may be performed on the gene:

1. *Insertion* adds an incoming connection with a strength randomly selected from $\{-1, 1\}$. The gene at the tail of the new connection is selected at random among the genes which do not yet have a connection from them to the current gene.

2. *Deletion* deletes a randomly selected incoming connection of the current gene by setting its weight to 0.

3. *Density-preserving mutation*, which is a deletion event followed by an insertion event.

The probability of density-preserving mutation $p_{dpm} = 0.5$ in all our experiments. Probabilities of insertion p_{ins} and deletion p_{del} are controlled using their ratio $r_{\text{insdel}} \equiv p_{\text{ins}}/p_{\text{del}}$. In all of our experiments this ratio was set to 1.

If any operation is impossible, e.g. if there is no incoming connections to this node and deletion is invoked, the node's incoming connections remain unchanged. Thus, density-preserving mutation only really preserves density when it is applied to a node with one or more incoming connections.

Initial populations: We consider two types of initial populations of the networks. We will say that an initial population is composed of *random* networks if the networks are generated by choosing connection strength from $\{-1, 0, 1\}$ at random for every possible connection in the network.

The alternative to this is to build the initial population out of randomly generated *sparse* networks. To obtain such a networks, we create a network without any

connections and mutate it once. In the resulting network every node has at most one incoming connection and possibly multiple outgoing connections.

Modularity metric: We quantify the modularity of evolved networks using the Q metric (e.g. [26, 36]). For a given decomposition of a network into modules it measures the difference between the actual fraction of the edges within modules and the expected fraction of edges for a random network of the same density. Q is defined to be the maximum value of such a difference across all possible decompositions of a network into modules. To find the optimal decomposition, we use Fast Unfolding of Communities method [10].

Density: The density of the network is defined as the number of connections in the network divided by the total number of possible connections, N^2.

Results

We investigated the performance of our multiobjective evolutionary algorithm (see Methods) under the following three sets of objectives and conditions:

P&CC-random This setup is similar to [26]. Following [26], the probability that the connection cost objective is taken into account was set to $p = 0.25$. Evolution starts with an initial population of random networks.

P&CC-sparse Same as P&CC-random, but the initial population is composed of randomly generated sparse networks.

P&TSM-sparse The two objectives of performance and TSM are taken into account deterministically ($p = 1$). Evolution starts with a population of sparse networks.

The performance of the approaches was measured using the task of finding a network with N nodes and N^2 possible connections that settles into the attractor in which neighboring gene values are maximally different:

$$\mathbf{s}' = (1 - 2(i \mod 2) \text{ for } i = 1, 2, ..., N).$$

Two variants of this task were considered: variant **A** with $N = 10$ and variant **B** with $N = 30$.

The comparison is presented in Fig. A.1. In both tasks, the P&CC-random approach led to slower adaptation than both the P&CC-sparse and P&TSM-sparse approaches. For task A, the average fitness for P&CC-random across 100 runs was significantly lower than the fitness for both P&CC-sparse ($p = 6 \cdot 10^{-5}$ with the Mann-Whitney U test implementation from scipy.stats) and P&TSM-sparse ($p < 2 \cdot 10^{-6}$) at generation 25. Later in the evolutionary history P&CC-random reaches the same values of fitness as P&TSM-sparse does, marginally surpassing the P&CC-sparse approach ($p < 3 \cdot 10^{-3}$ at generation 125).

For the more complex task B, the speedup caused by seeding the initial population with sparse networks is greater. Here, P&CC-random's fitness was worse than the fitness of the two other approaches throughout the whole run of 2250 generations ($p < 6 \cdot 10^{-14}$). The two approaches which start with populations of sparse networks – P&CC-sparse and P&TSM-sparse – show similar adaptation curves for both tasks, with P&TSM-sparse performing slightly better ($p < 4 \cdot 10^{-5}$ for generations 50-800; the highest ratio of the mean fitnesses is 1.054).

The patterns of variation of the Q value are different for all approaches. For both tasks the connection cost-based techniques both evolve networks with high Q which

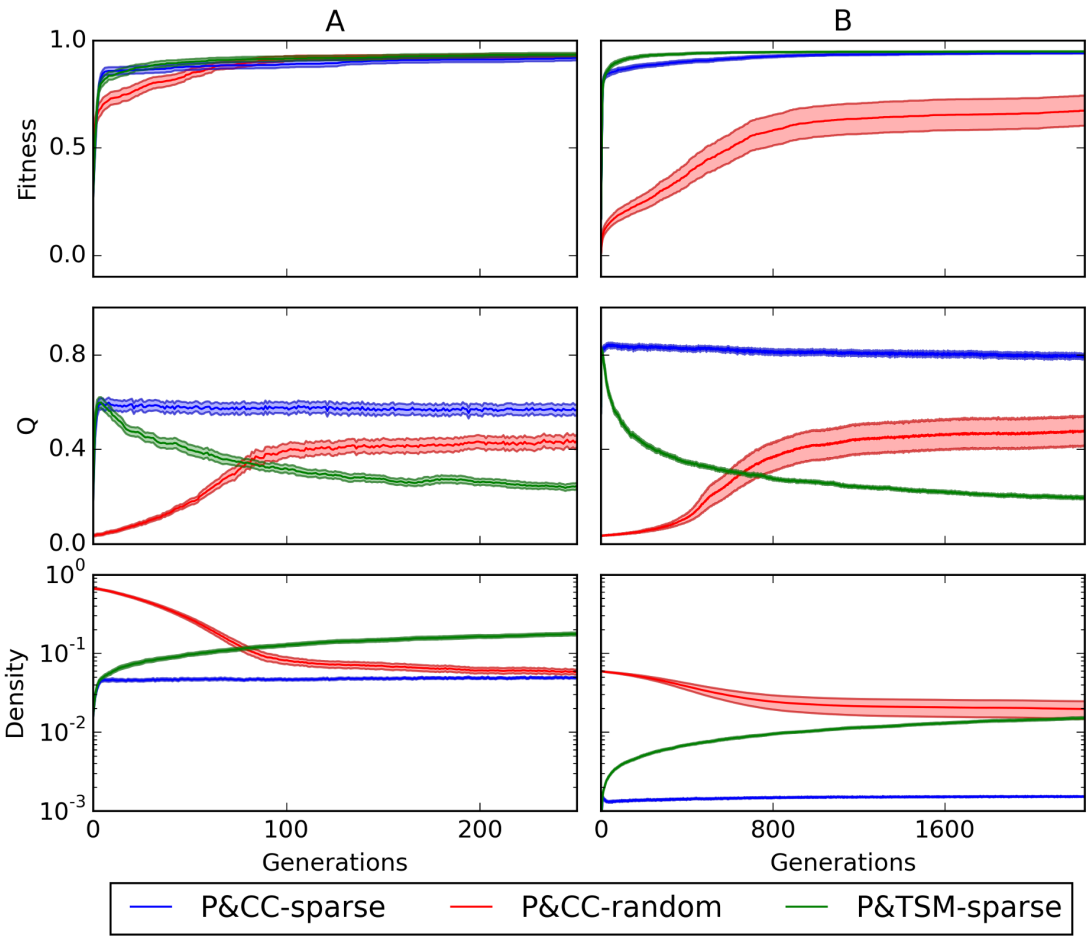


Figure A.1: Comparison of parameters of the most fit networks evolved with different approaches. Columns A and B correspond to tasks A and B in the text. The lines represent mean values over 100 runs; bands are 95% confidence intervals for Student t-distribution.

is maintained among the champions of subsequent generations. However, it takes less generations for P&CC-sparse than it does for P&CC-random. The number of the required generations has increased approximately ninefold for task B, which is in agreement with out $\mathcal{O}(N^2)$ hypothesis. The final Q value achieved by the P&CC-sparse approach is higher than that of P&CC-random for both tasks ($p < 4 \cdot 10^{-6}$).

For both tasks the Q metric of P&TSM-sparse follows the same rising pattern as it does for P&CC-sparse during the first few generations. Both approaches develop highly modular solutions at this point, but for subsequent generations modularity of P&TSM-sparse solutions falls rapidly while the modularity of P&CC-sparse solutions remains the same. This ultimately causes P&TSM-sparse to produce the least modular solutions for both tasks ($p < 9 \cdot 10^{-5}$).

The changes in density follow the changes in Q values. For task A, P&CC-random and P&CC-sparse stabilize at similarly low density, although it takes longer for P&CC-random to reach this state. For task B, the P&CC-random method evolves networks whose density stabilizes at a much higher value, perhaps due to the algorithm becoming trapped at local optima. For the P&TSM-sparse approach, the density keeps growing, but growth slows over generations.

Discussion

Our findings confirm that seeding evolution with a population of randomly generated sparse networks can facilitate the evolution of modularity and increase the rate of adaptation. In our experiments this approach worked better for when we evolved bigger networks.

We demonstrated that this effect is present for a biobjective performance plus

connection cost (P&CC) algorithm similar, but not identical, to the one described in [26]. However, when we seeded a multiobjective algorithm which minimized time since the last mutation instead of connection cost (P&TSM), we found that this initial increase in modularity decays over generations. Despite this, the new algorithm adapts approximately as fast as (and sometimes faster than) the P&CC-sparse algorithm for our task.

These results suggest that it is possible to replace the connection cost objective with another objective and still obtain, at equivalent evolutionary rate, networks of equivalent performance, possibly at the expense of some penalty to modularity. We speculate that replacement of connection cost with another diversity-promoting objective such as age [102] or novelty [82] may be beneficial for some harder tasks.